SocketDebugger

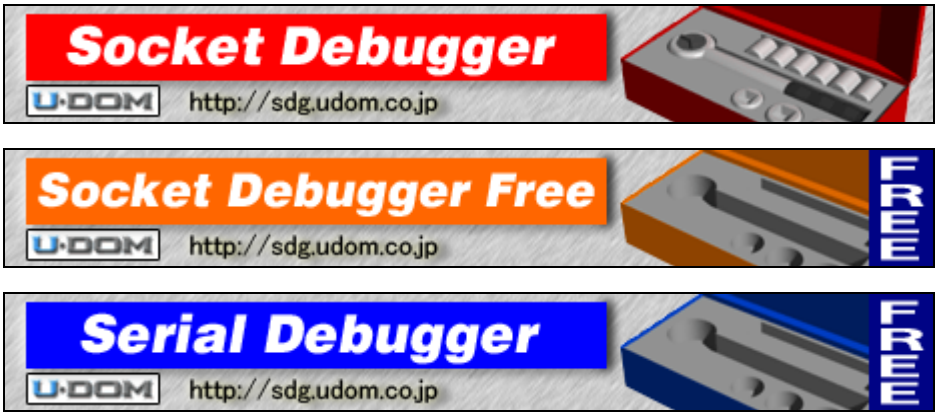
with

SocketDebuggerFree

SerialDebugger

Ver. 1.17

取扱説明書



株式会社ユードム

はじめに

この度は SocketDebugger のご選択、まことにありがとうございます。SocketDebugger はソケット通信、シリアル通信の統合テストツールです。本ツールを利用いただくことで、上記通信を行うシステム開発を強力にサポートします。是非、本マニュアルを手引きに SocketDebugger をフル活用いただけますよう、お願い申し上げます。

製品の構成

SocketDebugger はシェアウェアです。
他にフリー製品の下記を含み、シリーズ製品としています。
ソケット通信テスト用 SocketDebuggerFree
シリアル通信テスト用 SerialDebugger
（機能比較表はマニュアル中に記載します。）

SocketDebugger は上記２製品の全ての機能とプラスαの機能（パケットキャプチャ等）を有します。また、作成した通信設定を保存することにより効率よくテスト手法の管理を行うことが可能です。

フリー製品はそれ単体でご利用することも可能ですし、SocketDebugger 登録ユーザであれば、フリー製品向けの通信設定ファイルを作成し、フリー製品ユーザに通信の設定を与え、
通信シミュレーションの手順を示すことが可能です。
本機能を利用することで、登録ユーザは、例えば別な会社のソフト担当者などに
テストモジュールとして、通信の手順と効率のよいテストシミュレーションを
与えることが可能となります。

是非、この機会にフリーユーザのかたも、ご登録いただき、全ての機能の活用をご検討下さい。

ご注意
本マニュアル中に のアイコンが記載されています。
これらは各製品を表現しており、本アイコンにより製品ごとの処理の実装の違いを示します。
以下にアイコンと製品を説明します。

SocketDebugger　　 SocketDebuggerFree　　 SerialDebugger

本ソフトの特徴

●ソケット機能
・TCP クライアント、TCP サーバ、　UDP(UniCast,BroadCast,MultiCast)
・多彩なポート設定/自動動作設定
●シリアル機能
・詳細なフロー制御設定
・専用ウィンドウによる信号線状態設定&監視&中間バッファ処理
・多彩なポート設定/自動動作設定
●パケットモニタ機能
・IP 層でのパケットキャプチャ
・TCP,UDP,ICMP,IGMP ヘッダ分析機能
・分割パケットの組み立てサポート
・表示フィルタ、カラーフィルタ設定
・パケットデータの保存、読み込み
・パケットデータの再利用
●スクリプトによる制御拡張
・Lua スクリプトを記述することにより柔軟な通信制御を表現することが可能
●共通機能
・シリアル←→ソケットの関連処理が可能
・送信データ編集にバイナリエディタを実装。保存/読込 可能
・通信状態、送受信ログのリスト表示、詳細表示、保存、読込
・同時に２つのポートの実行が可能。複数ポートを使用するシステムに対応
・ドッキングウィンドウを採用した使いやすい GUI
・プラグイン開発による独自プロトコルの対応
・通信(動作)設定の保存/読込
・SocketDebuggerFree,SerialDebugger と併せて使用することにより
　通信設定ファイルを送付して
　通信シミュレーションを第三者に提供することが可能

免責事項

本ソフトウエアを使用することにより発生したいかなる損害も補償いたしません。
使用者の責任においてご使用されるようお願いいたします。

目次

1	**環境**		7
	1	動作環境	8
	2	インストール	9
	3	アンインストール	10
	4	関連ファイル	11
2	**ウィンドウ**		12
	1	ウィンドウ操作	13
	2	メインウィンドウ（メニュー）	17
	3	送信データエディタ	27
	4	詳細ログデータ	30
	5	接続状況	32
	6	送信データエクスプローラ	33
	7	COM ポート制御	34
	8	スクリプトエディタ	37
3	**通信設定**		38
	1	基本情報	39
	2	TCP クライアント通信	40
	3	TCP サーバ通信	42
	4	UDP 通信	44
	5	COM 通信	47
	6	動作設定	52
	7	動作拡張設定(拡張 DLL)	56
	8	ログ	58
	9	ソケットオプションの設定	60
4	**パケットキャプチャ**		62
	1	機能概略	63
	2	メインウィンドウ	65
	3	パケットヘッダ	71
	4	バイナリデータ	72
	5	キャプチャ設定	73
	6	表示フィルタ設定	74
	7	パケット編集	79

5	**ツール**		81
	1	その他の設定	82
	2	関連付け	87
	3	コマンド発行ツール	88
	4	ソケットエラー一覧	90
	5	COM ツール	91
	6	バージョン情報	92
	7	テキスト入力ダイアログ	93
	8	データ解析ダイアログ	94
6	**その他**		95
	1	機能比較表	96
	2	バイナリエディタ機能詳細	97
	3	拡張 DLL の作り方	101
	4	Lua スクリプト拡張説明	106
	5	Lua スクリプト命令拡張説明	116

環境

ここでは SocketDebugger の動作環境や、インストール、アンインストール、関連するファイルなどについて説明します。

１－１．動作環境

SocketDebugger（及び SocketDebuggerFree,SerialDebugger）は
下記の OS で動作確認を行っています。

Windows2000
WindowsXP

（以下でも、おおよその動作確認は行っています。
　一部、管理者権限を要求される機能があります。）
Windows Vista
Windows 7
Windows 8

※WindowsNT4.0 について
全ての機能を利用することができませんが、ある程度動作します。
以下、注意点とさせて下さい。
パケットキャプチャを行うことができません。
サービスパック、IE の状況により文字化けや IE コントロールの有無等あるため
全ての環境で正常動作できるものではありません。

動作 PC の推奨スペックなどは特に設定しておりませんが、
PC 性能により、動作時刻、パケットモニタの動作に差異が発生する場合が
ございます。

SocketDebugger は下記のツールで開発されています。
Visual Studio C++ 6.0、MFC スタティックリンク
Prof-UIS ライブラリ
Lua 5.1.2

１－２．インストール

SocketDebugger 及び SocketDebuggerFree,SerialDebugger 共に
特別なインストールは必要ありません。
解凍後、適当なフォルダに格納しご利用下さい。
設定ファイルなどと関連付け設定を行えるため格納フォルダを決めて
使用されることを推奨いたします。
関連付けの機能は SocketDebugger のみ可能です。

動作後は下記レジストリを使用します。
HKEY_CURRENT_USER¥Software¥exgroup

動作フォルダに設定ファイルを作成します。
default.sdc（パケットモニタフィルタファイル）

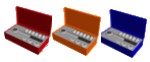
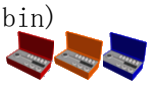
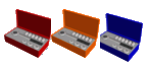
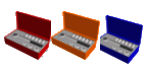
１－３．アンインストール

アンインストールについては exe ファイル、関連ファイルの削除を行ってください。
下記レジストリに情報が残りますのでそちらも削除してください。
HKEY_CURRENT_USER¥Software¥exgroup

関連付けを行った場合は下記レジストリも削除して下さい。
削除はアプリケーションの関連付けからも行えるのでそちらで削除すれば手動削除は不要です。
HKEY_CLASSES_ROOT¥.sdg
HKEY_CLASSES_ROOT¥.sdl
HKEY_CLASSES_ROOT¥SocketDebugger.Document
HKEY_CLASSES_ROOT¥SocketDebugger.Log

１－４．関連ファイル

SocketDebugger は下記のファイルを出力します。

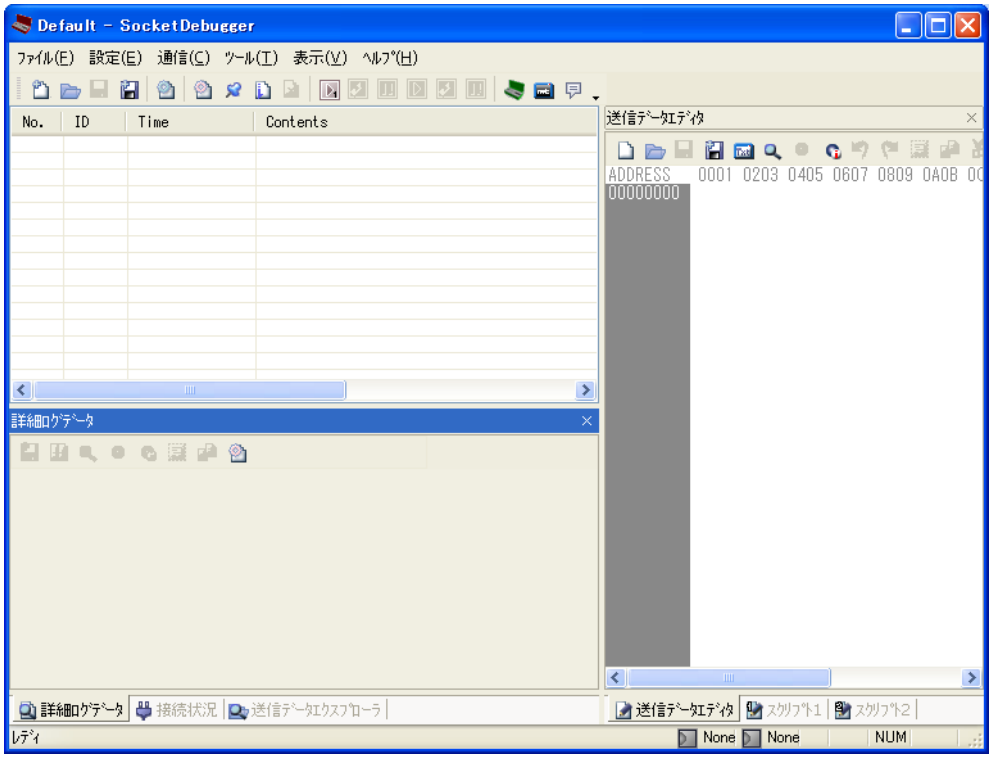
通信設定ファイル(.sdg)	通信の設定を保存したファイルです。通信設定のほか、送信データエディタ、Lua スクリプト、マーカー情報の内容についても保存されます。 アプリケーションが関連付けされていれば、設定ファイルからアプリケーションの起動が可能です。
auto.sdg	通信設定ファイルのうち、名称を「auto.sdg」としたものは自動通信開始用設定ファイルとなります。作成は SocketDebugger のみ行うことが可能です。 本ファイルと exe を同フォルダに配置し、exe を起動することにより通信設定が自動で読み込まれ、通信処理を開始します。 この機能を利用することにより、 SocketDebuggerFree、SerialDebugger についても通信設定を与えることが可能となります。
動作ログファイル(.sdl) 	通信の動作ログを保存したファイルです。 アプリケーションが関連付けされていれば、ログファイルからアプリケーションの起動が可能です。 メインウィンドウにドラッグドロップを行えます。 通常は任意で保存を行いますが、設定により自動保存が可能となります。 自動の場合は１時間に１ファイルで AUTOSAVEYYYYMMDDHH.sdl の名称形式で保存されます。 AUTOSAVE の部分を任意で設定可能です。
バイナリファイル(.bin) 	送信データ、受信データ保存時のデータファイルです。 名称、拡張子は任意ですが、デフォルトで.bin になっています。
Default.sdc	パケットモニタ機能の設定フィルタファイルです。
パケットログファイル(.sdp)	パケット(キャプチャ)ログファイルです。 任意で保存、読み込みを行うことが可能です。
パケットログファイル (Ethereal,libpcap)	Ethereal で作成されたパケットログファイルを読み込むことが可能です。
パケットフィルタファイル (. sdc)	パケットキャプチャ機能で、パケットの表示や色情報のフィルタ設定保存ファイルです。任意で保存、読み込みを行うことが可能です。
マーカーファイル (.mrk 	送信データエディタ、詳細ログデータのマーカー情報 （背景色、コメントなど）を記録したファイルです。
スクリプトファイル (.lua) 	通信のスクリプト制御に記述したプログラムファイルです。 任意にファイル保存可能です。 SocketDebugger では、sdg ファイルに自動で保存されます。

ウィンドウ

ここでは、各ウィンドウ、メニューの使い方などについて
説明します。

２－１．ウィンドウ操作

初期ウィンドウ表示時は以下のように表示されます。
ここでは既にサブウィンドウ（６枚）が張り付いた状態となっています。



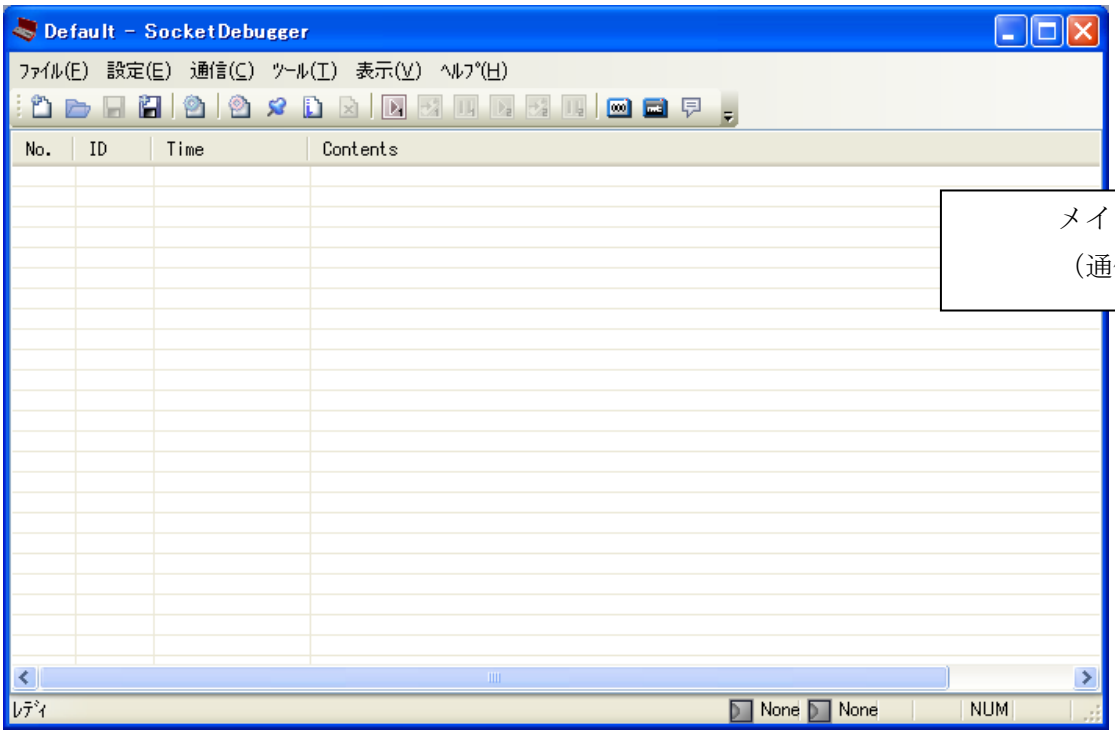
通常はこのようなウィンドウで操作を行いますが、サブウィンドウはドッキングウィンドウとなっているので、
自由な位置に貼り付け、タブ化、または独立ウィンドウ化を行うことができます。
位置を変える場合は各ウィンドウのウィンドウバーをマウスで掴んで移動させて下さい。
タブ化してある場合は、タブを掴んで移動させて下さい。
標準ではユーザーインタフェース（UI）が「Visual Stadio 2005」となっていますので
移動中は貼り付けガイドが表示されます。希望の位置のアイコン上にマウスを移動することで
ウィンドウの貼り付けが行えます。
別な UI の場合は貼り付けたい位置にマウスを移動することで貼り付けイメージが表示されるので、
希望の位置でマウスを離して下さい。

（UI の切替は SocketDebugger のみの機能です。）

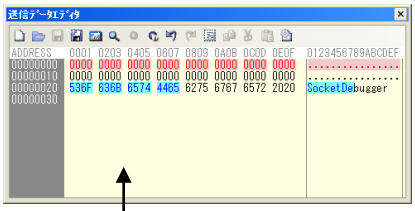
サブウィンドウの表示／非表示を切り替えるにはメニューの「表示」から選択します。
非表示はウィンドウの×ボタンでも可能です。

貼り付けガイド
上下左右と中心のタブ化を指定できる。

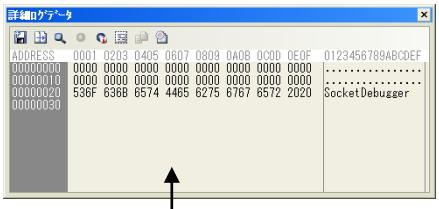
メインウィンドウ、サブウィンドウは下記のような構成となっています。



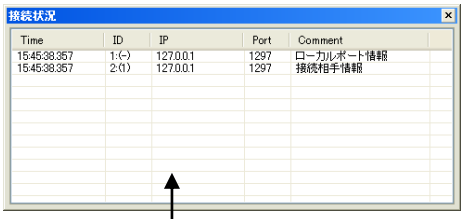
メインウィンドウ
（通信ログ表示）



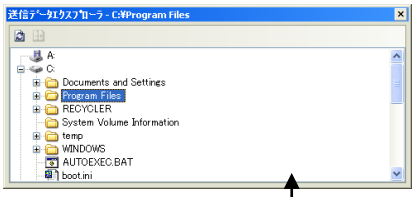
サブウィンドウ１
（送信データエディタ）



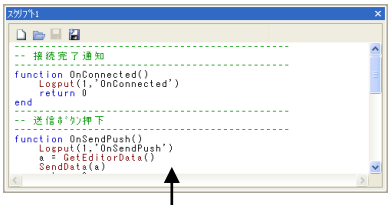
サブウィンドウ２
（詳細ログデータ）



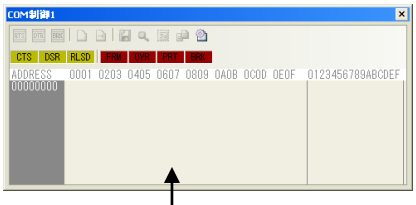
サブウィンドウ３
（接続状況）



サブウィンドウ４
（送信データエクスプローラ）

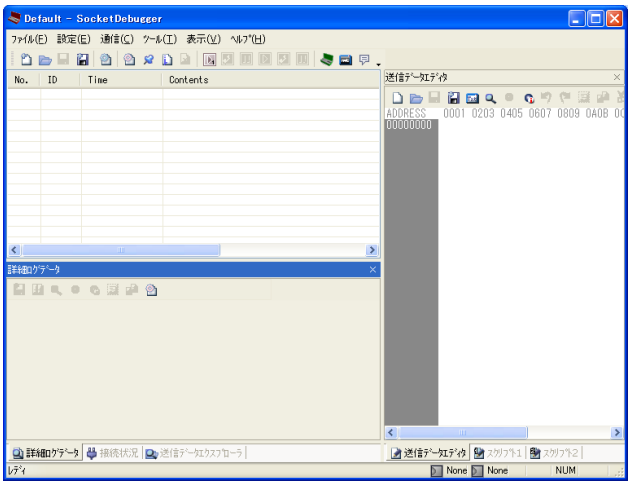


サブウィンドウ５
（スクリプト１）
サブウィンドウ６
（スクリプト２）



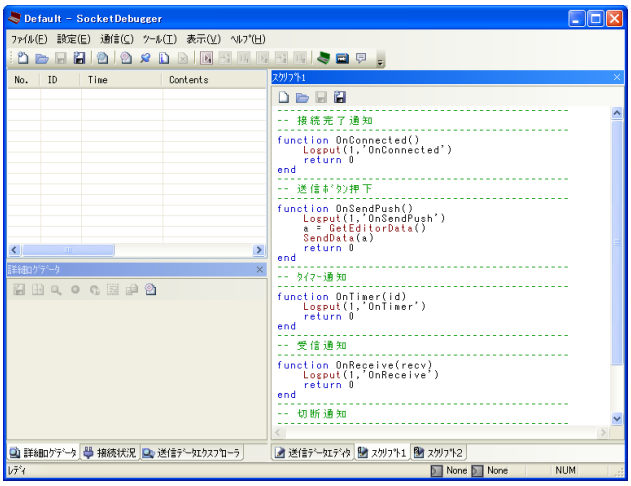
サブウィンドウ７
（COM 制御１）
サブウィンドウ８
（COM 制御２）

各サブウィンドウのドッキング例を紹介します。



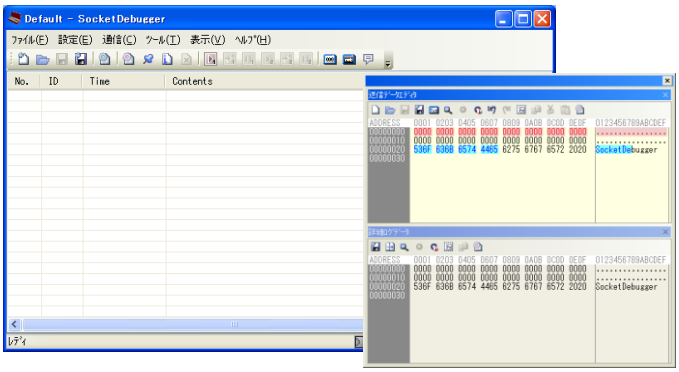
1．ベーシック

デフォルトの形式です。右に送信データエディタ、スクリプトエディタを設置してデータ、制御の入力を行い、上部のツールバーで通信を制御します。画面左上のメイン部でログを確認しつつ、左下の詳細ログデータで通信の詳細確認を行います。状況にあわせて接続状況の確認や送信データエディタにセットするファイルを送信データエクスプローラで選択します。



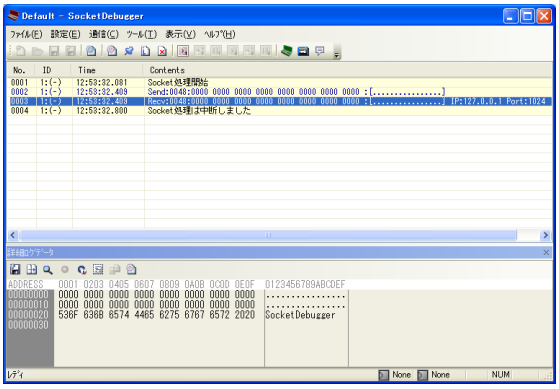
2．スクリプト重視

スクリプトエディタを表示して、通信を試しながら処理を記述します。



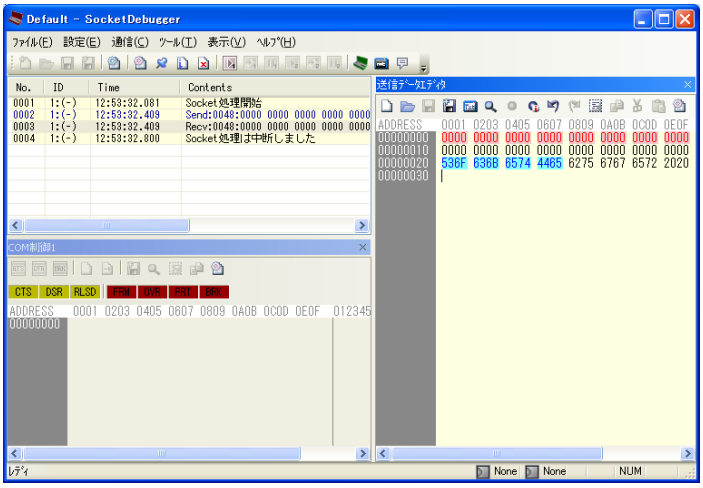
3．分離

ログ部分と送信データエディタ＋詳細ログデータウィンドウを分離して操作します。



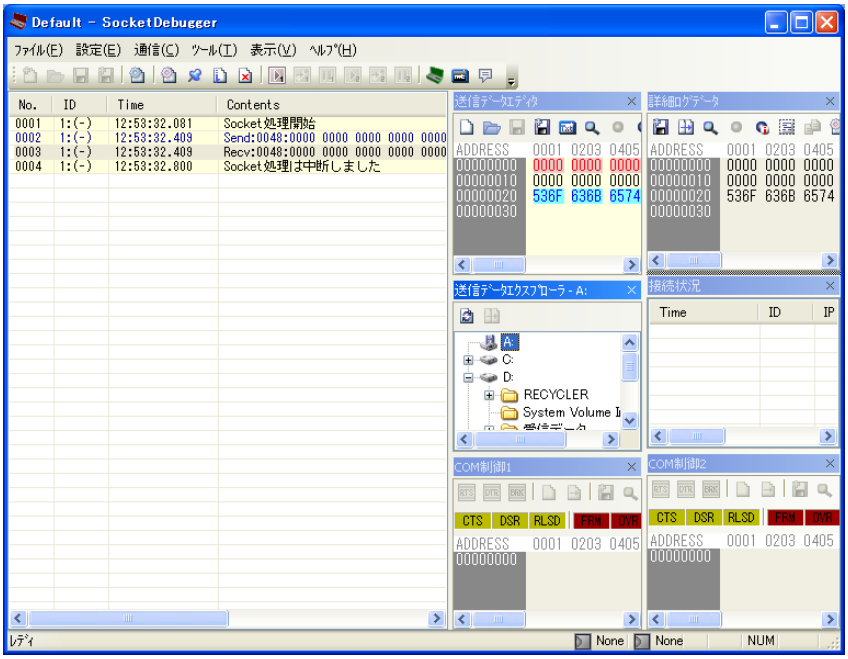
4．ログ重視

ログ部分を左右フルに利用し、画面下に詳細ログデータのみ表示しログの確認だけを行います。通常はアイコン化して画面から消して他アプリケーションを操作するのに向いています。



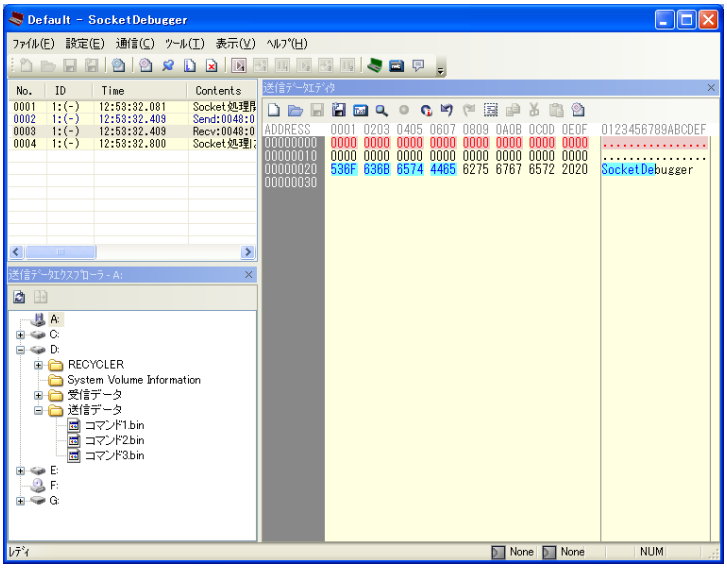
5．シリアル制御

右の送信データエディタで送信データを編集し、左下に配したCOM制御でシリアルの制御処理を行います。



6．ALL

全てのサブウィンドウを表示し、状況の総合判断を行います。

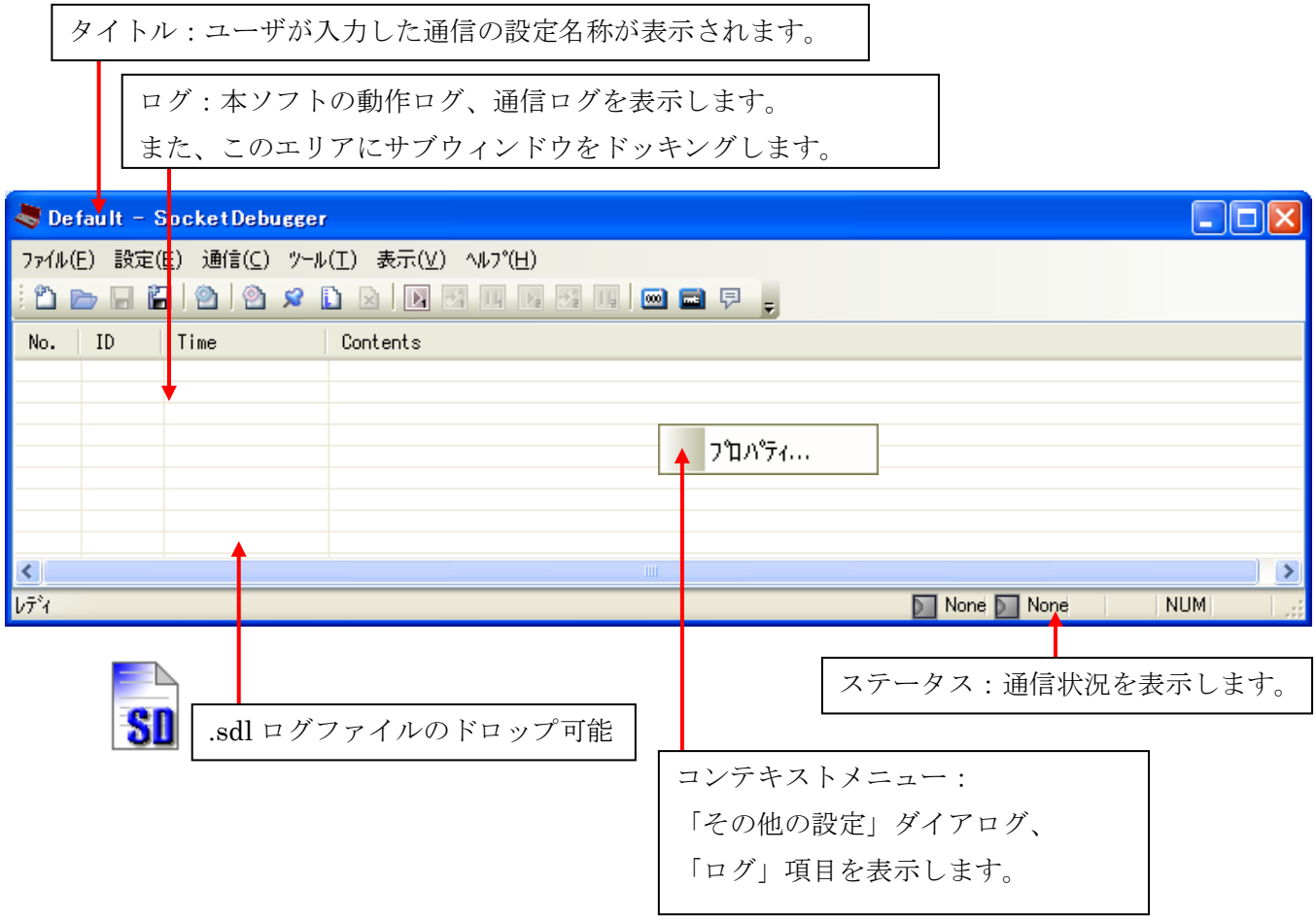


7．バイナリエディタ

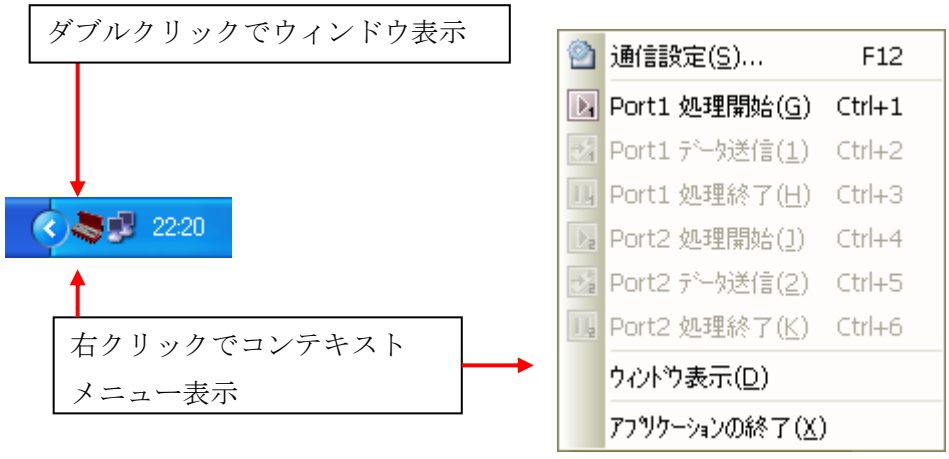
送信データエクスプローラと送信エディタで便利なバイナリファイルエディタとなります。ファイルを左ペインから簡単に選択できるのでスピーディーにファイル編集が可能です。

２－２．メインウィンドウ

ログ表示機能を持ったメインウィンドウの説明を行います。
メインウィンドウは、メニュー、ツールバー、ステータスバーを含んでおり、本ソフトの総合的な操作を行います。



設定により、タスクトレイにアイコンを表示することが可能です。
（設定は「５－１．その他の設定」を参照下さい。）



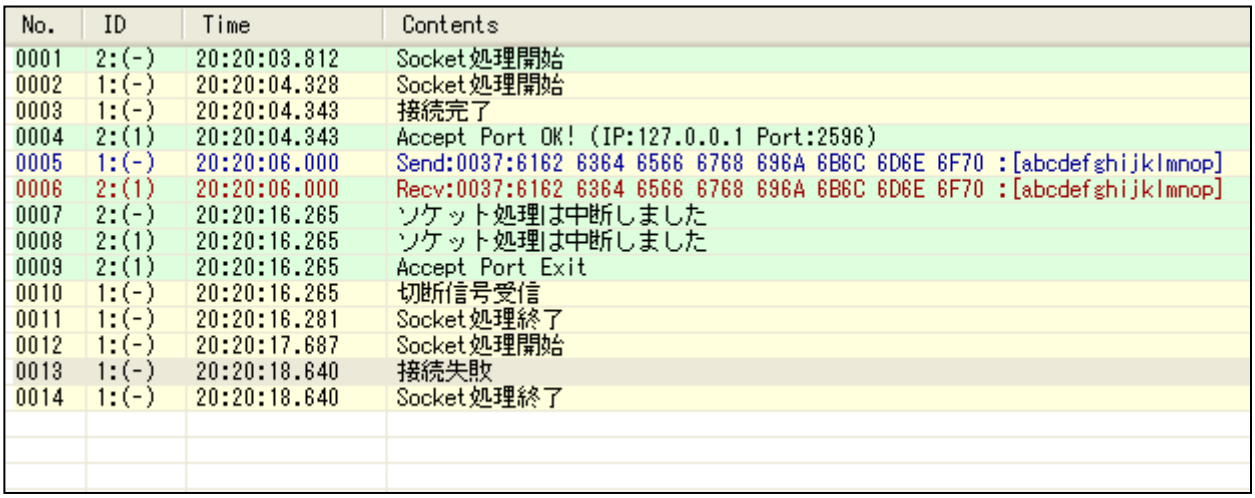
ログ表示

通信処理において発生した状況を発生順に下に追加表示します。

ログをクリックすることで、その詳細状態を「詳細ﾛｸﾞﾃﾞｰﾀ」ウィンドウに表示します。

また、ログの表示の可/否、表示色、表示項目内容について、設定を行うことが可能です。

詳細は「３－８．ログ」「５－１．その他の設定」を参照下さい。



No.	ID	Time	Contents
0001	2:(-)	20:20:03.812	Socket処理開始
0002	1:(-)	20:20:04.328	Socket処理開始
0003	1:(-)	20:20:04.343	接続完了
0004	2:(1)	20:20:04.343	Accept Port OK! (IP:127.0.0.1 Port:2596)
0005	1:(-)	20:20:06.000	Send:0037:6162 6364 6566 6768 696A 6B6C 6D6E 6F70 :[abcdefghijklmnop]
0006	2:(1)	20:20:06.000	Recv:0037:6162 6364 6566 6768 696A 6B6C 6D6E 6F70 :[abcdefghijklmnop]
0007	2:(-)	20:20:16.265	ソケット処理は中断しました
0008	2:(1)	20:20:16.265	ソケット処理は中断しました
0009	2:(1)	20:20:16.265	Accept Port Exit
0010	1:(-)	20:20:16.265	切断信号受信
0011	1:(-)	20:20:16.281	Socket処理終了
0012	1:(-)	20:20:17.687	Socket処理開始
0013	1:(-)	20:20:18.640	接続失敗
0014	1:(-)	20:20:18.640	Socket処理終了

No	ログ番号
ID	通信設定のポートを示します。 通常、先頭の数値の設定番号のみが有効データですが サーバ接続の場合は（）内の数値が接続しているソケットの接続番号となります。
Time	事象の発生した時刻です。
Contents	ログのダイジェストです。詳細ログデータウィンドウで全ての内容の閲覧が可能です。 ログには大きく、２種類存在します。 ○　動作ログ 通信の開始、終了、接続、切断やエラー情報などの文字による通信動作における状況についての文字メッセージ。 ○　通信ログ 送信したデータ、受信したデータの記録。 下記フォーマットにて表示を行います。 [送信 or 受信],[データサイズ], [先頭 16 バイトバイナリダイジェスト], [先頭 16 バイトアスキーダイジェスト] [（UDP の受信データの場合）送信元 IP,ポート表示]

メニュー解説

「ファイル」

新規作成	
通信設定を初期状態に戻し、新たにユーザが設定する準備を行います。通信設定名称は default になります。実行前に現在の設定を破棄する確認ウィンドウが表示されます。通信処理中は選択できません。	
開く	
通信設定ファイルの選択ダイアログを表示し、保存されている設定ファイルの読み込みを行います。通信設定に「読み込み時に自動で処理を開始する」の属性がついている場合は読み込み後、すぐに通信処理が開始されます。通信処理中は選択できません。確認ウィンドウは表示されませんのでご注意下さい。	
保存	
現在の通信設定を上書き保存します。確認がありませんのでご注意下さい。「開く」「名前を付けて保存」など、ファイル名称が確定している場合に選択可能となります。	
名前を付けて保存	
現在の通信設定を名前を付けて保存します。名称入力ダイアログが表示されます。このとき、ファイル名称を「auto.sdg」とすることで、自動開始用通信設定ファイルとなります。 詳細は「１－４．関連ファイル」を参照下さい。	
ﾛｸﾞﾌｧｲﾙの読込	
動作ログファイル選択ダイアログを表示し、保存されているログファイルの読み込みを行います。既に表示中のログは削除されます。 メインウィンドウに動作ログファイルをドロップすることでログファイルを読み込むことも可能です。	
ﾛｸﾞﾌｧｲﾙの保存	
メインウィンドウで管理している動作ログに名前を付けて保存します。名称入力ダイアログが表示されます。	
ﾛｸﾞをﾃｷｽﾄで出力	
メインウィンドウで管理している動作ログを **CSV、またはテキスト**ファイルに保存します。**CSV、またはテキストの選択ダイアログと**名称入力ダイアログが表示されます。**CSV はログのダイジェスト、テキストは全通信データのレポート保存です。**	
ｱﾌﾟﾘｹｰｼｮﾝの終了	
アプリケーションを終了します。確認ウィンドウは表示されませんのでご注意下さい。	

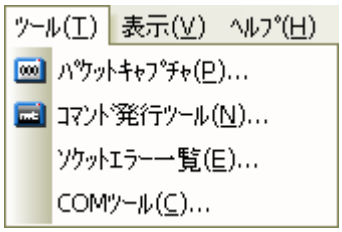
「設定」

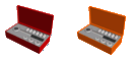
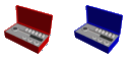
通信設定
通信の設定ダイアログを表示します。 詳細は「３．通信設定」の項を参照下さい。
その他の設定
本ソフトの環境的な設定を行います。設定はアプリケーション終了時にレジストリに保存され、以後、起動時に読み込まれます。 詳細は「５－１．その他の設定」の項を参照下さい。
最前面に表示
メインウィンドウの最前面表示切替を行います。最前面に表示中はチェックアイコンが表示されます。チェックを外せば元の状態に戻ります。
自動ｽｸﾛｰﾙ
ログにデータが追加された際に自動で、「最新データを選択」「可視位置までのスクロール」を行うか指定します。指定中はチェックアイコンが表示されます。チェックを外せば元の状態に戻ります。 ※指定中はログデータが自動で選択、詳細ログデータへの表示となるため頻繁にデータが追加になる場合、負荷が高くなる可能性があります。
ﾛｸﾞ削除
現在管理されているメインウィンドウのログ情報を削除します。 確認ダイアログが表示されます。
関連付け
関連付けダイアログが表示され、通信設定ファイル(.sdg)と動作ログファイル(.sdl)のレジストリによる関連付け（また、その解除）を行います。 関連付けを行うと上記ファイルにアイコンが設定され、ファイルからアプリケーションの起動を行うことが可能となります。 詳細は「５－２．関連付け」の項を参照下さい。

「通信」

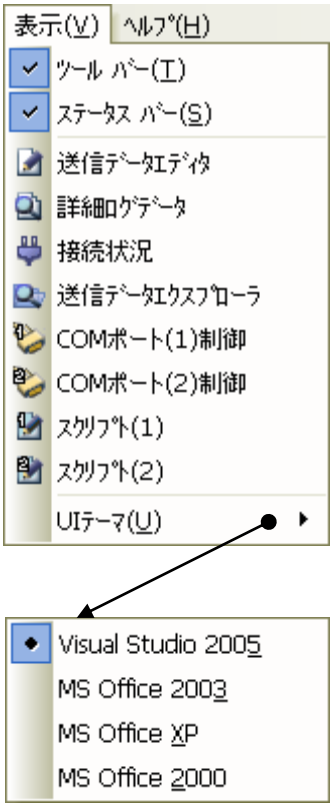
Port1 処理開始	
Port1 に設定された通信処理を開始します。 通信中、及び設定が無い場合は選択できません。	
Port1 データ送信	
Port1 の設定で通信中の相手に「送信データエディタ」の内容を(手動)送信します。Port1 に対して複数の通信相手が存在した場合、その全てに送信します。 通信処理停止状態、及び TCP の場合、未コネクションの場合は選択できません。	
Port1 処理終了	
Port1 に対して通信処理の(手動)終了を行います。 複数のコネクションが存在した場合も一括で切断を行います。 通信が未開始の場合は選択できません。	
Port2 処理開始	
Port2 の設定を対象に通信の開始を行います。 詳細は Port1 を参照下さい。	
Port2 データ送信	
Port2 の設定を対象にデータの送信を行います。 詳細は Port1 を参照下さい。	
Port2 処理終了	
Port2 の設定を対象に通信の終了を行います。 詳細は Port1 を参照下さい。	

「ツール」



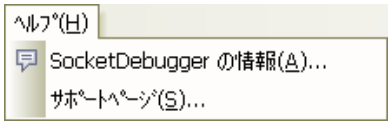
ﾊﾟｹｯﾄｷｬﾌﾟﾁｬ	
パケットキャプチャウィンドウを表示します。 詳細は「４．パケットキャプチャ」の項を参照下さい。	
ｺﾏﾝﾄﾞ発行ﾂｰﾙ	
コマンド発行ツールダイアログを表示します。 詳細は「５－３．コマンド発行ツール」の項を参照下さい。	
ｿｹｯﾄｴﾗｰ一覧	
ソケットエラー一覧ダイアログを表示します。 詳細は「５－４．ソケットエラー一覧」の項を参照下さい。	
COM ﾂｰﾙ	
COM ツールダイアログを表示します。 詳細は「５－５．COM ツール」の項を参照下さい。	

「表示」



ﾂｰﾙﾊﾞｰ	
ﾂｰﾙﾊﾞｰの表示/非表示を切り替えます。	
ｽﾃｰﾀｽﾊﾞｰ	
ｽﾃｰﾀｽﾊﾞｰの表示/非表示を切り替えます。	
送信ﾃﾞｰﾀｴﾃﾞｨﾀ	
(ｻﾌﾞｳｨﾝﾄﾞｳ)送信ﾃﾞｰﾀｴﾃﾞｨﾀの表示/非表示を切り替えます。	
詳細ﾛｸﾞﾃﾞｰﾀ	
(ｻﾌﾞｳｨﾝﾄﾞｳ)詳細ﾛｸﾞﾃﾞｰﾀの表示/非表示を切り替えます。	
接続状況	
(ｻﾌﾞｳｨﾝﾄﾞｳ)接続状況の表示/非表示を切り替えます。	
送信ﾃﾞｰﾀｴｸｽﾌﾟﾛｰﾗ	
(ｻﾌﾞｳｨﾝﾄﾞｳ)送信ﾃﾞｰﾀｴｸｽﾌﾟﾛｰﾗの表示/非表示を切り替えます。	
COMﾎﾟｰﾄ(1)制御	
(ｻﾌﾞｳｨﾝﾄﾞｳ) COMﾎﾟｰﾄ(1)制御の表示/非表示を切り替えます。	
COMﾎﾟｰﾄ(2)制御	
(ｻﾌﾞｳｨﾝﾄﾞｳ) COMﾎﾟｰﾄ(2)制御の表示/非表示を切り替えます。	
ｽｸﾘﾌﾟﾄ(1)	
(ｻﾌﾞｳｨﾝﾄﾞｳ) ﾎﾟｰﾄ(1)に対するｽｸﾘﾌﾟﾄｴﾃﾞｨﾀの表示/非表示を切り替えます。	
ｽｸﾘﾌﾟﾄ(2)	
(ｻﾌﾞｳｨﾝﾄﾞｳ) ﾎﾟｰﾄ(2)に対するｽｸﾘﾌﾟﾄｴﾃﾞｨﾀの表示/非表示を切り替えます。	
UI ﾃｰﾏ	
ウィンドウユーザインタフェースのイメージを切り替えます。 下記は各表示サンプルです。	
Visual Studio 2005 	MS Office 2003
MS Office XP 	MS Office 2000

「ヘルプ」



SocketDebugger の情報	
各アプリケーションのバージョンを示すダイアログを表示します。 詳細は「５－６．バージョン情報」の項を参照下さい。	
ｻﾎﾟｰﾄﾍﾟｰｼﾞ	
サポート WEB ページをブラウザを開いて表示します。	

「タスクトレイメニュー」

通信設定
通信の設定ダイアログを表示します。 詳細は「３．通信設定」の項を参照下さい。
Port1 処理開始
Port1 に設定された通信処理を開始します。 通信中、及び設定が無い場合は選択できません。
Port1ﾃﾞｰﾀ送信
Port1 の設定で通信中の相手に「送信データエディタ」の内容を(手動)送信します。**Port1** に対して複数の通信相手が存在した場合、その全てに送信します。 通信処理停止状態、及び **TCP の場合、**未コネクションの場合は選択できません。
Port1 処理終了
Port1 に対して通信処理の(手動)終了を行います。 複数のコネクションが存在した場合も一括で切断を行います。 通信が未開始の場合は選択できません。
Port2 処理開始
Port2 の設定を対象に通信の開始を行います。 詳細は **Port1** を参照下さい。
Port2ﾃﾞｰﾀ送信
Port2 の設定を対象にデータの送信を行います。 詳細は **Port1** を参照下さい。
Port2 処理終了
Port2 の設定を対象に通信の終了を行います。 詳細は **Port1** を参照下さい。
ｳｨﾝﾄﾞｳ表示
メインウィンドウを表示します。
ｱﾌﾟﾘｹｰｼｮﾝの終了
アプリケーションを終了します。確認ウィンドウは表示されませんのでご注意下さい。

ステータスバー解説

メニューや、ツールバー選択中にマウス下の項目の説明が表示されます。

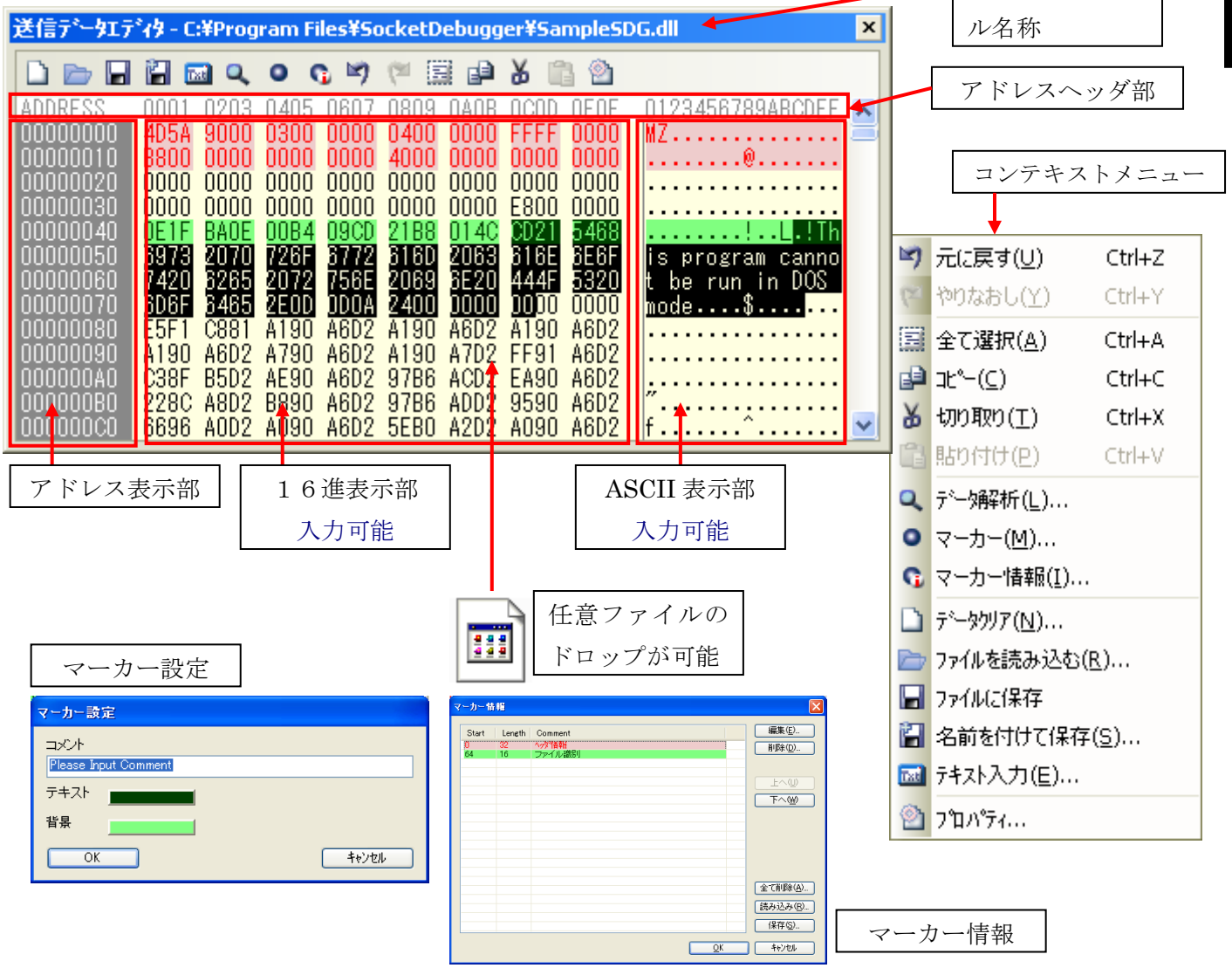
左側:Port1、右側:Port2 の通信状況を表示します。

通信状況アイコンの説明

アイコン	TCP クライアント	TCP サーバ	UDP	COM
通信処理なし	通信処理未開始	通信処理未開始	通信処理未開始	通信処理未開始
リトライ待機中	リトライ開始まで待機中 （この状況は通信処理の開始は行っているが、内部的に待機状態となっている）		リトライ開始まで待機中 （この状況は通信処理の開始は行っているが、内部的に待機状態となっている）	リトライ開始まで待機中 （この状況は通信処理の開始は行っているが、内部的に待機状態となっている）
接続要求中	コネクション処理の実行中			
接続中	コネクションが確立している		通信処理開始中	通信処理開始中
接続受付中（現在接続無し）		接続受付中であるが、接続ｿｹｯﾄ無し （右側の数字は接続中のｿｹｯﾄの数）		
接続受付中（現在接続有り）		接続受付中で、接続ｿｹｯﾄ有り （右側の数字は接続中のｿｹｯﾄの数）		

２－３．送信データエディタ

サブウィンドウ：送信データエディタでは、送信するデータ（および受信応答するデータ）を
バイナリエディタ機能を使って編集します。



●入力は１６進表示部と ASCII 表示部のどちらからでも可能です。
●アンドゥ・リドゥが可能です。
●作成したデータはファイルに保存することが可能です。
●ファイルを選択し、読み込むことが可能です。（ファイルのドロップにも対応しています。）
●テキスト入力ダイアログ、バイナリデータ解析ダイアログサポート。
●データにマーカーを設定することが可能です。
　マーカーはテキスト色、背景色、ツールチップコメントの設定を行えます。
●表示のデザインについて「その他の設定ダイアログ：送信データエディタ」
　（または本ウィンドウの「プロパティ」）から変更が可能です。使い易い設定でお使い下さい。
●バイナリエディタ機能についての詳細は「６－２．バイナリエディタ機能詳細」を参照下さい。

メニュー解説（１）

ﾃﾞｰﾀｸﾘｱ	現在のデータ、UNDO 履歴、ファイルとの関係を削除します。 確認ダイアログが表示されます。
ﾌｧｲﾙを読み込む	ファイルを選択して読み込みます。 ファイル選択ダイアログが表示されます。
ﾌｧｲﾙに保存	現在のエディタの内容を現在関連づけられているファイル（タイトルバーに表示されているファイル名称）に上書き保存します。 ファイルが関連付けられていない場合は選択できません。
名前を付けて保存	現在のエディタの内容をファイル名称を指定して保存します。名称入力ダイアログが表示されます。
ﾃｷｽﾄ入力	テキスト専用入力ダイアログを表示します。送信データエディタの内容がテキスト表現可能な場合にご利用いただくとメモ帳イメージでの入力が行うことが可能です。詳細は「５－７．テキスト入力ダイアログ」の項を参照下さい。
ﾃﾞｰﾀ解析	カーソル位置からのバイナリ情報を WORD、DWORD、float、double 型などの情報として解析を行うダイアログを表示します。 詳細は「５－８．データ解析ダイアログ」の項を参照下さい。
マーカー	現在選択しているエリアにマーカーを設定します。 テキスト色、背景色、ツールチップコメントの設定ダイアログが表示されます。 既にマーカー設定されている箇所を含む場合は削除の確認ダイアログを表示します。 マーカー機能の詳細は「６－２．バイナリエディタ機能詳細」を参照下さい。
マーカー情報	現在設定されているマーカー情報の一覧を表示します。 テキスト色、背景色、ツールチップコメントの変更と、マーカー情報ファイルへの保存、読み込みなどが可能です。

メニュー解説（２）

元に戻す 	（UNDO）行ったエディタ操作を取り消しひとつ前の状態に戻します。
やりなおし 	（RED）UNDO 操作を取り消してひとつ前の状態に戻します。
全て選択 	現在の全バイナリデータを選択状態にします。
ｺﾋﾟｰ 	現在の選択範囲をクリップボードにコピーします。
切り取り 	現在の選択範囲をクリップボードにコピーして、エディタからは削除します。
貼り付け 	現在のクリップボードの内容をカーソル位置（または選択範囲）に挿入（上書き）します。
ﾌﾟﾛﾊﾟﾃｨ 	表示デザインを設定するダイアログを表示します。 詳細は「５－１．その他の設定」の項を参照下さい。

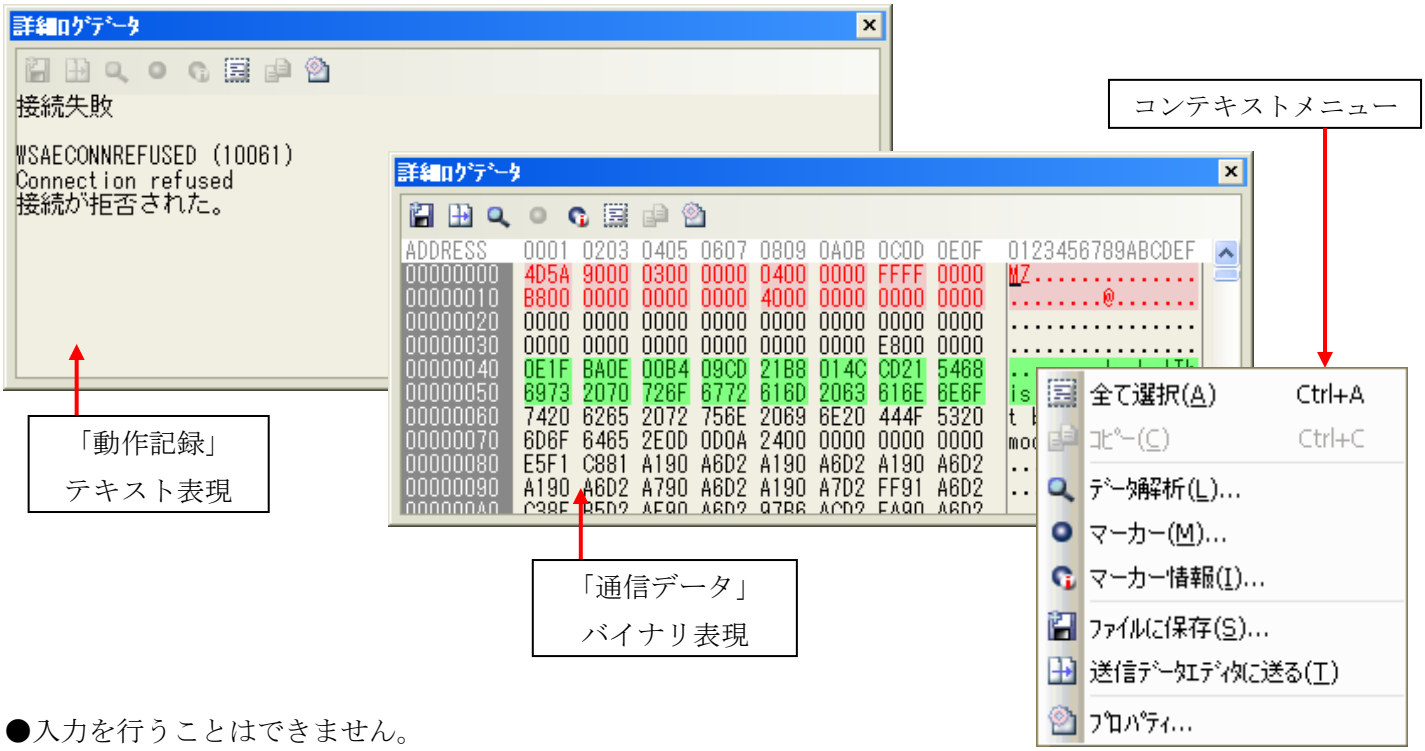
２－４．詳細ログデータ

サブウィンドウ：詳細ログデータでは、メインウィンドウで選択された通信ログに関する詳細情報を表示します。ログは「動作記録」「通信データ」の２つに分類されます。

●「動作記録」の場合、テキスト表現

●「通信データ」の場合、バイナリ表現

選択されたログにより、表示内容が自動で切り替わります。



●入力を行うことはできません。

●通信データを送信データエディタにセットできます。

●バイナリデータ解析ダイアログサポート。

●データにマーカーを設定することが可能です。

　マーカーはテキスト色、背景色、ツールチップコメントの設定を行えます。

●表示のデザインについて「その他の設定ダイアログ：詳細ログデータ」

　（または本ウィンドウの「プロパティ」）から変更が可能です。使い易い設定でお使い下さい。

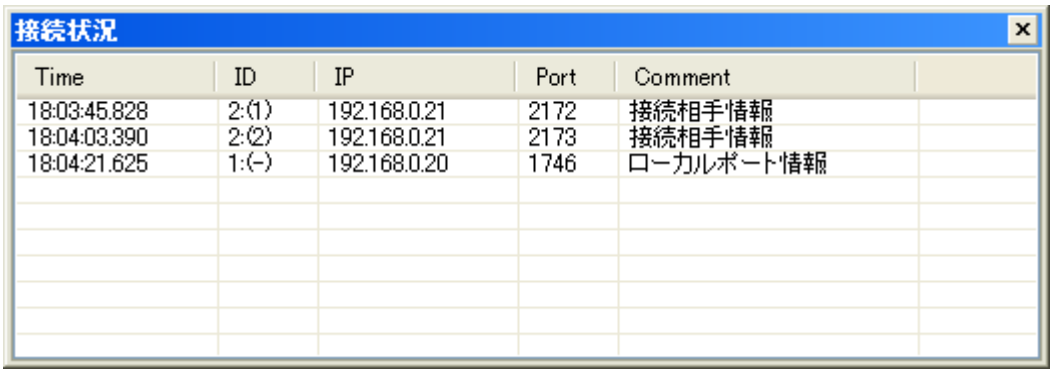
●バイナリエディタ機能についての詳細は「６－２．バイナリエディタ機能詳細」を参照下さい。

メニュー解説

ﾌｧｲﾙに保存	現在のデータ内容をファイル名称を指定して保存します。 名称入力ダイアログが表示されます。 通信データ表示中のみ選択可能です。
送信ﾃﾞｰﾀｴﾃﾞｨﾀに送る	現在のデータ内容を送信データエディタにセットします。 通信データ表示中のみ選択可能です。
ﾃﾞｰﾀ解析	カーソル位置からのバイナリ情報を WORD、DWORD、float、double 型などの情報として解析を行うダイアログを表示します。 詳細は「５－８．データ解析ダイアログ」の項を参照下さい。 通信データ表示中のみ選択可能です。
マーカー	現在選択しているエリアにマーカーを設定します。 テキスト色、背景色、ツールチップコメントの設定ダイアログが表示されます。 既にマーカー設定されている箇所を含む場合は削除の確認ダイアログを表示します。 マーカー機能の詳細は「６－２．バイナリエディタ機能詳細」を参照下さい。 通信データ表示中のみ選択可能です。
マーカー情報	現在設定されているマーカー情報の一覧を表示します。 テキスト色、背景色、ツールチップコメントの変更と、マーカー情報ファイルへの保存、読み込みなどが可能です。 通信データ表示中のみ選択可能です。
全て選択	現在の表示データを選択状態にします。
ｺﾋﾟｰ	現在の選択範囲をクリップボードにコピーします。
ﾌﾟﾛﾊﾟﾃｨ	表示デザインを設定するダイアログを表示します。 詳細は「５－１．その他の設定」の項を参照下さい。

２－５．接続状況

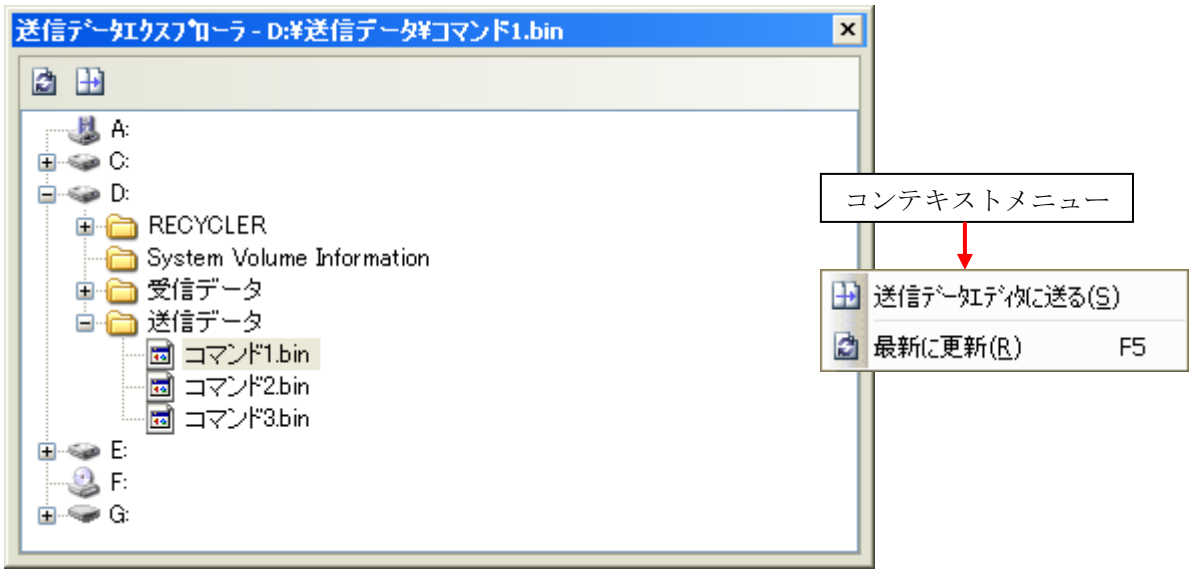
サブウィンドウ：接続状況では、現在通信中のポート情報を表示します。
表示内容は通信タイプにより異なります。
情報はリアルタイムで更新されます。



通信タイプ	Time	ID	IP	Port	Comment
TCP ｸﾗｲｱﾝﾄ	通信確立時刻	通信設定 Port 番号	自 PC の使用 IP	自 PC の 使用ﾎﾟｰﾄ番号	ローカルポート情報
TCP ｻｰﾊﾞ	通信確立時刻	通信設定 Port 番号と 接続ｿｹｯﾄ ID(接続順に 1 から割り当てされる。ｻｰﾊﾞ設定時は複数のｿｹｯﾄと接続可能なため)	接続している 相手 PC の IP	接続している 相手 PC ﾎﾟｰﾄ番号	接続相手情報
UDP	通信開始時刻	通信設定 Port 番号	送信相手の IP	送信相手のﾎﾟｰﾄ番号	送信先アドレス
COM	通信開始時刻	通信設定 Port 番号			

2－6．送信データエクスプローラ

サブウィンドウ：送信データエクスプローラでは、送信データエディタにセットするファイルを
エクスプローライメージで表示、選択します。
送信するデータパターンのファイルを作成しておくことで送信データエディタの内容を
簡単に切り替えることができます。

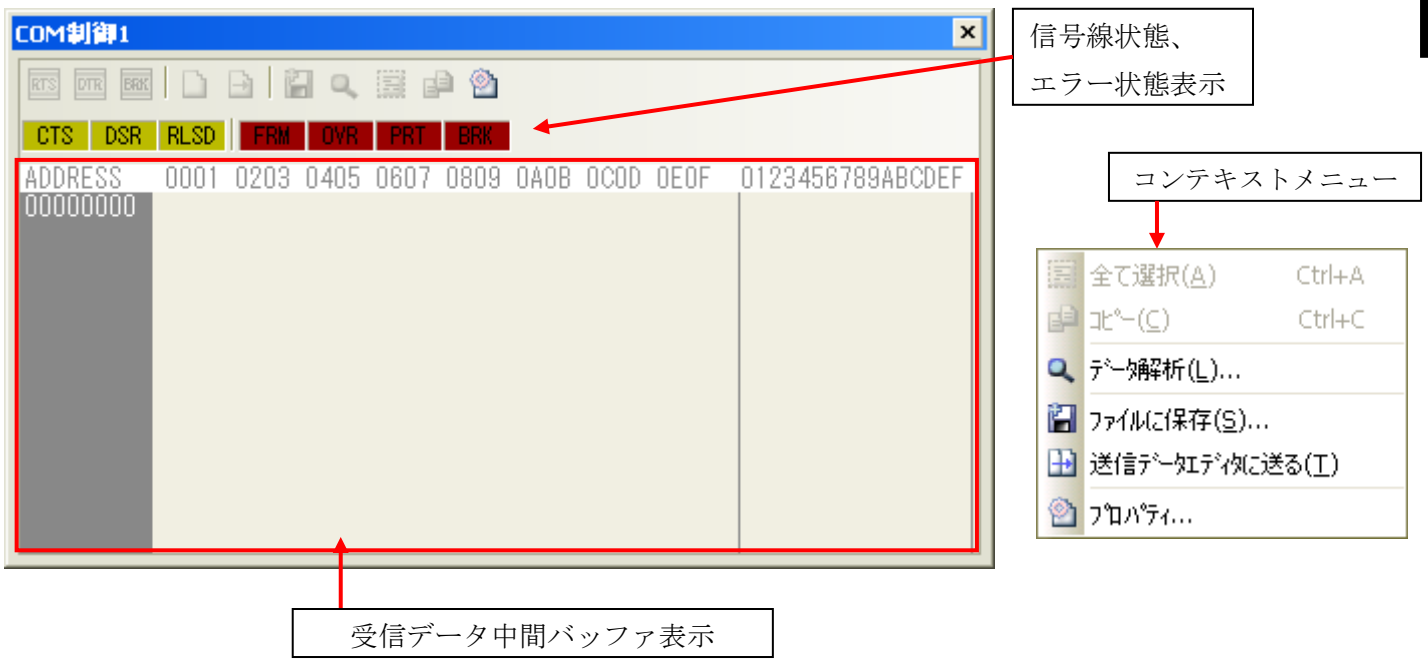


メニュー解説

最新に更新	開いているフォルダ内のファイルリストを再読み込みします。 本エクスプローラは自動更新されませんのでファイル構成が変更された 場合は本処理を行って下さい。
送信ﾃﾞｰﾀｴﾃﾞｨﾀに送る	選択されたファイルの内容を送信データエディタにセットします。 選択された項目がファイルの場合のみ選択可能です。

２－７．ＣＯＭポート制御

サブウィンドウ：ＣＯＭポート制御では、通信中のＣＯＭポートの操作支援制御を行います。
本ウィンドウは通信設定の Port1、Port2 でそれぞれ存在します。



●信号線状態(RTS, DTR)と BREAK の ON/OFF 指示を行えます。
●信号線状態(CTS, DSR, RLSD)の状態をリアルタイムで表示します。
●エラー状態(フレームエラー、オーバーラン、パリティエラー、ブレーク検出)をリアルタイムで表示します。
●受信データ中間バッファとは通信設定においてデータ区切りを固定サイズ、または特定 Char とした場合に、受信データの単位として扱われる前の受信データを表示します。
（生データと設定した場合は利用せずに受信したデータをそのまま受信データとして処理を行います。）
本ウィンドウから、中間バッファのクリアと、現時点のデータを受信データとして受け入れる操作を行えます。

メニュー解説

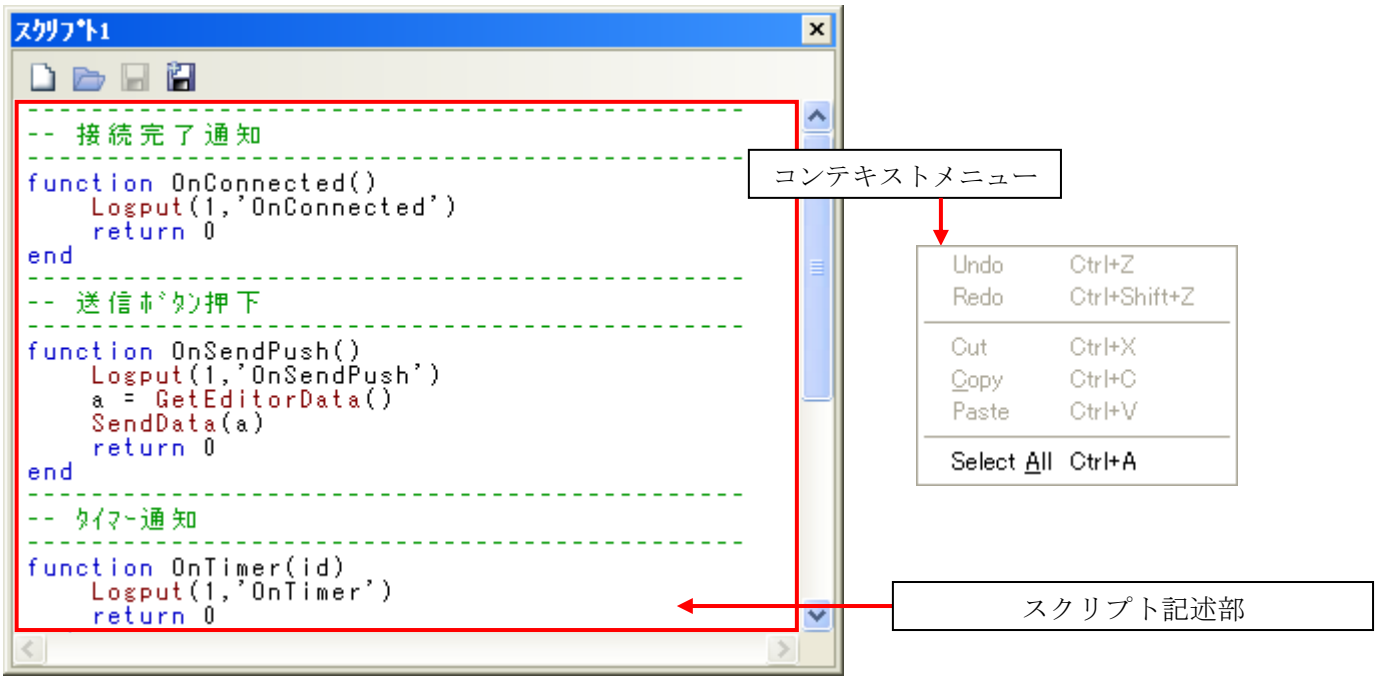
DTR 制御 	DTR 信号線の状態を ON/OFF します。 チェック状態で ON、未チェック状態で OFF を表現します。 通信中のみ選択できます。
RTS 制御 	RTS 信号線の状態を ON/OFF します。 チェック状態で ON、未チェック状態で OFF を表現します。 通信中のみ選択できます。
Break 制御 	Break 状態を ON/OFF します。 チェック状態で ON、未チェック状態で OFF を表現します。 通信中のみ選択できます。
ﾃﾞｰﾀｸﾘｱ 	現在の中間バッファの内容をクリアします。
ﾃﾞｰﾀ吐出 	現在の中間バッファの内容を受信データとして強制的に処理を行います。 中間バッファの内容はクリアされます。
送信ﾃﾞｰﾀｴﾃﾞｨﾀに送る 	現在の中間バッファの内容を送信データエディタにセットします。
名前を付けて保存 	現在の中間バッファの内容をファイル名称を指定して保存します。名称入力ダイアログが表示されます。
ﾃﾞｰﾀ解析 	カーソル位置からのバイナリ情報を WORD、DWORD、float、double 型などの情報として解析を行うダイアログを表示します。 詳細は「５－８．データ解析ダイアログ」の項を参照下さい。
全て選択 	現在の中間バッファデータ全てを選択状態にします。
ｺﾋﾟｰ 	現在の選択範囲をクリップボードにコピーします。
ﾌﾟﾛﾊﾟﾃｨ 	表示デザインを設定するダイアログを表示します。 詳細は「５－１．その他の設定」の項を参照下さい。

状態表示解説

CTS	CTS 信号線状態です。 「CTS（ 送信可）信号の状態が変わりました。」 ON でランプ点灯、OFF で消灯です。
DSR	DSR 信号線状態です。 「DSR（ データセットレディ）信号の状態が変わりました。」 ON でランプ点灯、OFF で消灯です。
RLSD	RLSD 信号線状態です。 「RLSD（ 受信線信号検出）信号の状態が変わりました。」 ON でランプ点灯、OFF で消灯です。
FRM	フレーミングエラー。 「ハードウェアがフレーミングエラーを検出しました。」 発生でランプ点灯、回復で消灯です。
OVR	オーバーランエラー状態です。 「文字バッファがいっぱいになりました。次の文字は失われます。」 発生でランプ点灯、回復で消灯です。
PRT	パリティエラー状態です。 「ハードウェアがパリティエラーを検出しました。」 発生でランプ点灯、回復で消灯です。
BRK	ブレーク検出状態です。 「ハードウェアがブレーク条件を検出しました。」 発生でランプ点灯、回復で消灯です。

２－８．スクリプトエディタ

サブウィンドウ：スクリプトエディタでは、Lua スクリプトを記述します。
スクリプトを作成することにより、通信ポートの動作をユーザが幅広い動作を指定することが可能です。
本ウィンドウは通信設定の Port1、Port2 でそれぞれ存在します。



●Lua スクリプトの詳細については、**６－４．Lua スクリプト**拡張説明を参照ください。

通信設定

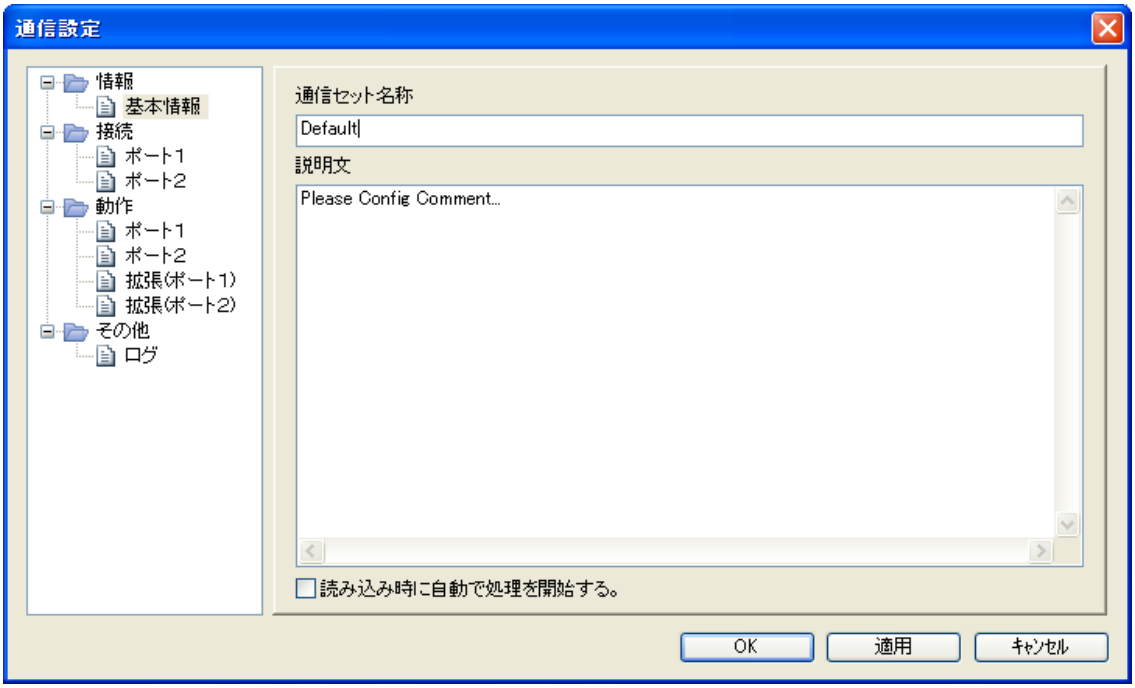
通信を行うための接続設定、動作設定について説明します。
ここで設定した内容は、ファイルに保存することが可能ですので複雑な設定も１度行えば再度設定する必要ありません。

通信設定の保存は、ここで設定する内容以外に、送信データエディタの内容、マーカー情報についても保存されます。

※ファイルの保存は SocketDebugger のみとなります。

３－１．基本情報

これから行う通信設定の任意情報を入力します。（入力は必須ではありません）
左側の項目ペインから基本情報を選択します。

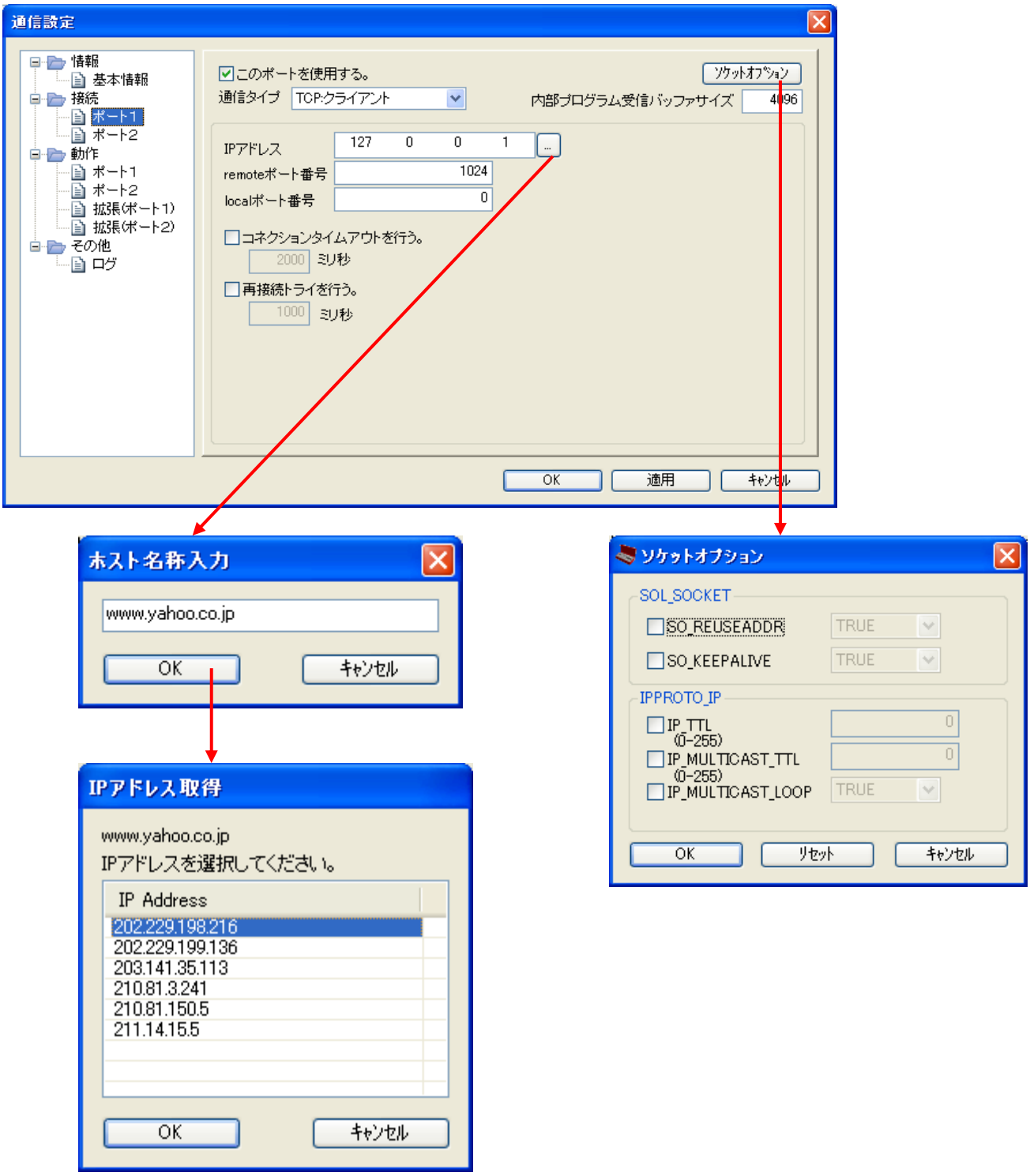


項目解説

通信セット名称	設定する通信情報に名称を付けます。入力した情報はメインウィンドウのタイトル部（および、タスクトレイツールチップ）に表示されるので、複数の通信を行う場合のウィンドウの判別に有効です。
説明文	設定した通信の条件や仕様などのメモ書きを保存できます。
読み込み時に自動で処理を開始する。	チェックを行うことで、通信設定ファイルを読み込んだ際に自動で通信処理を開始させることが可能です。 (SocketDebugger のみ)

３－２．ＴＣＰクライアント通信

TCP クライアントポートとして通信を行う場合の設定について説明します。
左側の項目ペインからポート１（または２）を選択します。
次に「このポートを使用する」にチェックを入れ、通信タイプに「TCP クライアント」を選択します。



項目解説

IP アドレス	通信相手装置のＩＰアドレスを入力します。 右の「…」ボタンを押して、ホスト名称から自動入力も可能です。 入力したホストが複数のＩＰアドレスを保持していた場合、 ＩＰアドレスリストが表示されますので任意のＩＰアドレスを選択して下さい。 ※デフォルトでは[127.0.0.1]が入力されています。これは自端末(localhost)を意味します。
remote ポート番号	通信相手装置の待機ポート番号を入力します。
local ポート番号	自端末の使用するポート番号を入力します。 通常は０で設定します。０の場合、利用するポートを装置が自動で 空きポートから割り当てます。 逆に指定を行ってしまうと再接続において TIMEWAIT 状態解消までポートが 利用できなくなってしまいますのでご注意下さい。 これはＴＣＰ／ＩＰの仕様です。
コネクションタイムアウトを行う	相手装置がネットワーク上見つからない場合、通常では接続できないことを 判定するまでに時間がかかります。 この設定を有効にすることで、アプリケーション的に接続できないと 判定するまでの待機時間を設定することが可能です。単位はミリ秒です。
再接続トライを行う	通信が切断した場合や、エラーが発生して通信処理が終了した場合に 自動で再接続を行うか設定できます。有効にした場合、 前回の通信処理が終了してから入力した時間を待機後、 再度接続処理を実行します。
内部プログラム受信バッファサイズ	通信実行時に、内部処理が使用する受信バッファのサイズを設定します。 このサイズにより１回の受信イベントで受信できる最大データ量が決まります。 （このサイズを超えるデータも TCP では分割して受信します。大きく確保しても受信タイミングごとに処理を行うため分割が発生しなくなる訳ではありません）
ｿｹｯﾄｵﾌﾟｼｮﾝ	ソケットオプションの設定を行うことが可能です。 詳細は「３－９．ソケットオプションの設定」を参照下さい。

通信手順

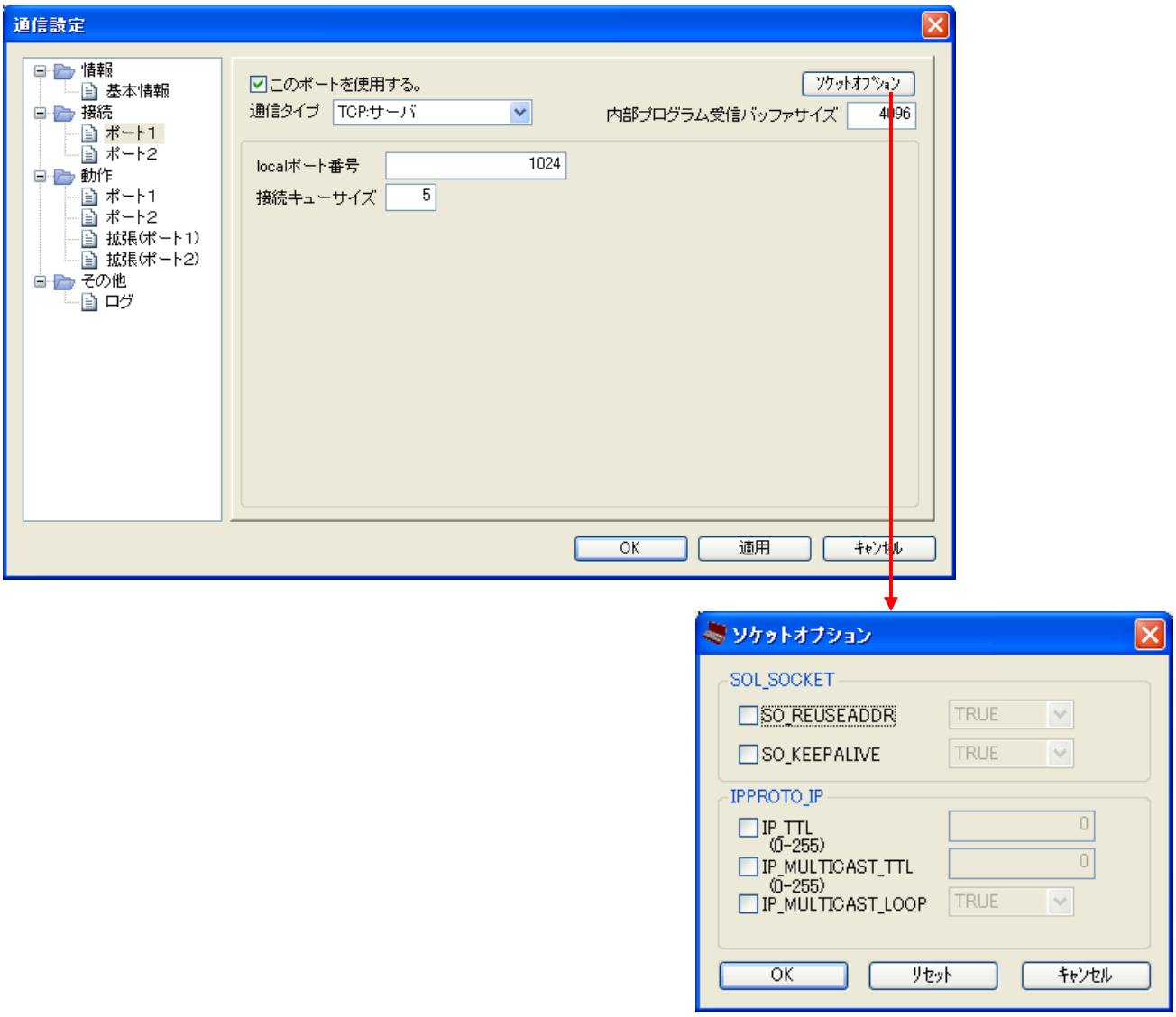
設定後、Port1(または 2)処理開始を選択します。接続要求が開始され、接続成功/失敗がログに表示されます。
接続に成功すると「データ送信」、「処理終了」が選択できるようになります。

自動処理の設定があれば、設定に従い処理を行いますが、
送信データエディタのデータを手動で「データ送信」や、任意での切断「処理終了」を行うことが可能です。

３－３．ＴＣＰサーバ通信

TCP サーバポートとして通信を行う場合の設定について説明します。
左側の項目ペインからポート１（または２）を選択します。
次に「このポートを使用する」にチェックを入れ、通信タイプに「TCP サーバ」を選択します。

TCP サーバでは、自端末の指定したポートで、他端末からの接続（TCP クライアントポートから）を待機します。接続後は、要求したポートに対して対応するポートを生成して処理を実行させるかたわら待機していたポートは継続して接続待機を行っているため、次の要求を受け付けて結果、複数の通信ポート処理を生成することになります。



項目解説

local ポート番号	接続待機に利用する自ポート番号を入力します。
接続キューサイズ	接続要求→接続許可（ACCEPT）までに待機させる要求の最大数を指定します。 ※接続可能なポートの数ではありませんので注意してください。
内部プログラム受信バッファサイズ	通信実行時に、内部処理が使用する受信バッファのサイズを設定します。 このサイズにより１回の受信イベントで受信できる最大データ量が決まります。 （接続されたポートごとに領域を確保します。） （このサイズを超えるデータも TCP では分割して受信します。大きく確保しても受信タイミングごとに処理を行うため分割が発生しなくなる訳ではありません）
ｿｹｯﾄｵﾌﾟｼｮﾝ	ソケットオプションの設定を行うことが可能です。 詳細は「３－９．ソケットオプションの設定」を参照下さい。

通信手順

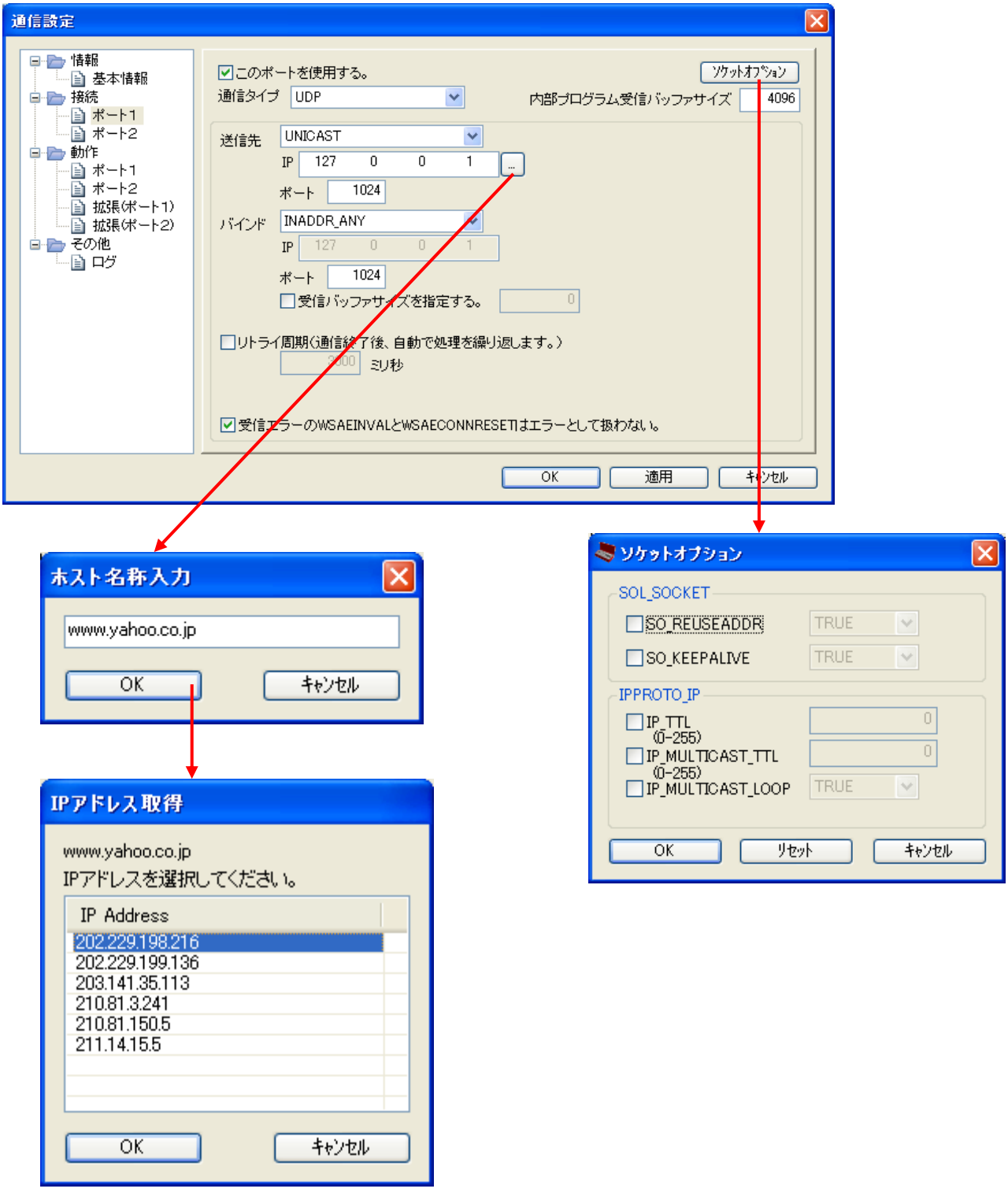
設定後、Port1(または 2)処理開始を選択します。
指定したポートに問題が無ければ接続待機が開始されます。
この状態では「処理終了」は選択できますが、「データ送信」はまだ選択できません。

クライアントポートの接続要求が発生すると自動で要求を受け付け、対応ポートを生成し、
「データ送信」が可能となります。

自動処理の設定があれば、設定に従い処理を行いますが、
送信データエディタのデータを手動で「データ送信」や、任意での切断「処理終了」を行うことが可能です。
「データ送信」を選択した場合は接続中の全ポートへ送信データエディタの内容を送信します。
「処理終了」を選択した場合は接続中の全ポートを切断し、受付待機ポートの受付停止を行います。

３－４．ＵＤＰ通信

UDP ポートとして通信を行う場合の設定について説明します。
左側の項目ペインからポート１（または２）を選択します。
次に「このポートを使用する」にチェックを入れ、通信タイプに「UDP」を選択します。



項目解説

送信先	送信先とするタイプを下記から選択します。 **UNICAST**:送信する装置を単一で選択します。 **BROADCAST**:セグメント内の全接続装置を送信先とします。 **MULTICAST**:マルチキャスト IP を送信先とします。
(送信先)IP	送信先の IP アドレスを入力します。 **UNICAST** 時は送信したい装置の IP アドレスを入力します。右の「…」ボタンからホスト名称による自動入力も可能です。 **BROADCAST** 時は入力不可となります。 **MULTICAST** 時は送信するマルチキャスト IP アドレスを入力します。 マルチキャスト IP とは通常[224.000.000.000]からのアドレスです。
(送信先)ポート	送信するポート番号を入力します。
バインド	自装置が自ポートをどのように扱うか設定します。 **IN_ADDRANY**:自装置の指定したポート番号に到着した通信データを認識できるように指定ポートをバインドします。 **バインドしない**:ポートのバインドを行わないため、受信処理を行いません。送信専用ポートとなります。 **IP アドレス指定**:自装置が複数の IP アドレスを保持していた場合に指定された IP､指定されたポートに到着した通信データのみを認識できるように指定ポートに IP アドレスを指定してバインドします。
(バインド)IP	バインドを行う IP アドレスを指定します。 バインド指定が **IP アドレス指定**の場合のみ入力可能です。
(バインド)ポート	バインドを行うポートを指定します。 **IN_ADDRANY** または **IP アドレス指定**の場合のみ入力可能です。
(バインド)受信バッファサイズを指定する	setsockopt 関数で SO_RCVBUF の指定を行います。 **IN_ADDRANY** または **IP アドレス指定**の場合のみ入力可能です。
リトライ周期	通信エラーなどが発生して処理が終了した場合に設定された時間後に自動で処理を再開します。 単位はミリ秒です。
受信エラーの WSAINVAL と WSAECONNRESET はエラーとして扱わない	データの送信時に相手 PC の当該ポートが受信待機していなかった場合などに、リセット信号が返り、受信エラーが発生してしまう場合があります。そういった場合にこの２つのエラーは無視して処理を継続するように設定します。 （ログは出力されますが、エラーとして終了はしません。）
内部プログラム受信バッファサイズ	通信実行時に、内部処理が使用する受信バッファのサイズを設定します。 このサイズにより受信できる最大データ量が決まります。 （UDP では、このサイズを超えるデータを受信するとエラーとなります。）
ｿｹｯﾄｵﾌﾟｼｮﾝ	ソケットオプションの設定を行うことが可能です。 詳細は「３－９．ソケットオプションの設定」を参照下さい。

通信手順

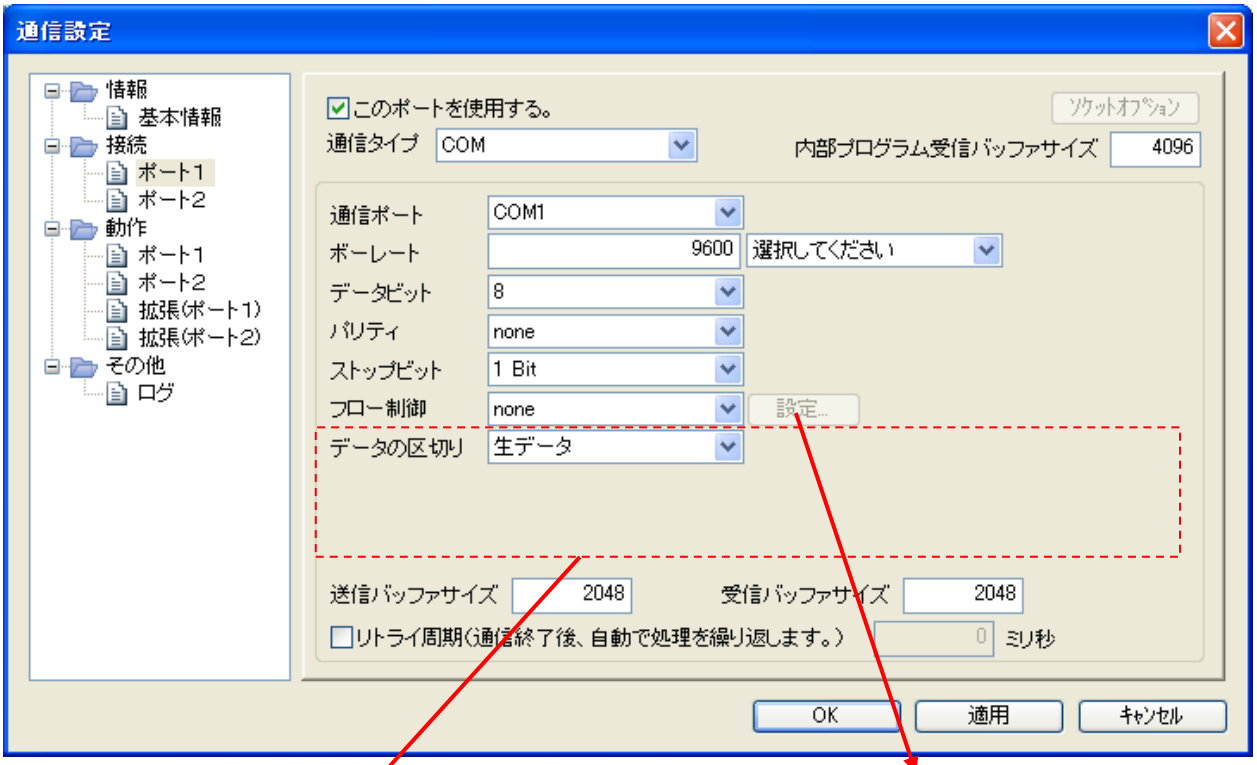
設定後、Port1(または 2)処理開始を選択します。
指定したポートに問題が無ければ処理が開始されます。
この状態で「データ送信」「処理終了」が選択可能となります。

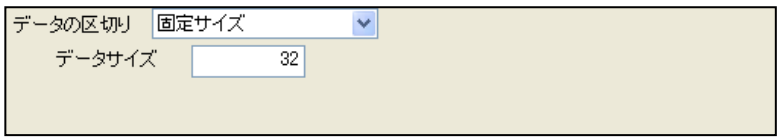
自動処理の設定があれば、設定に従い処理を行いますが、
送信データエディタのデータを手動で「データ送信」や、任意で「処理終了」を行うことが可能です。

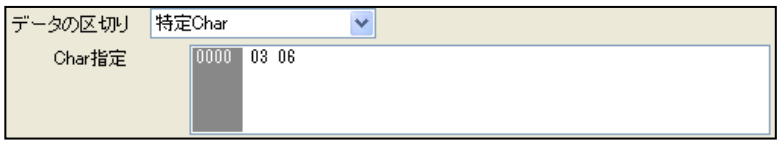
※１　通常、ブロードキャストでは MTU を超えるデータの送信は可能ですが受信ができないようです。
※２　UDP の場合、受信するデータのサイズが大きい場合は内部プログラム受信バッファサイズの
　　設定を行って下さい。設定が誤っている場合、受信エラーに直結します。

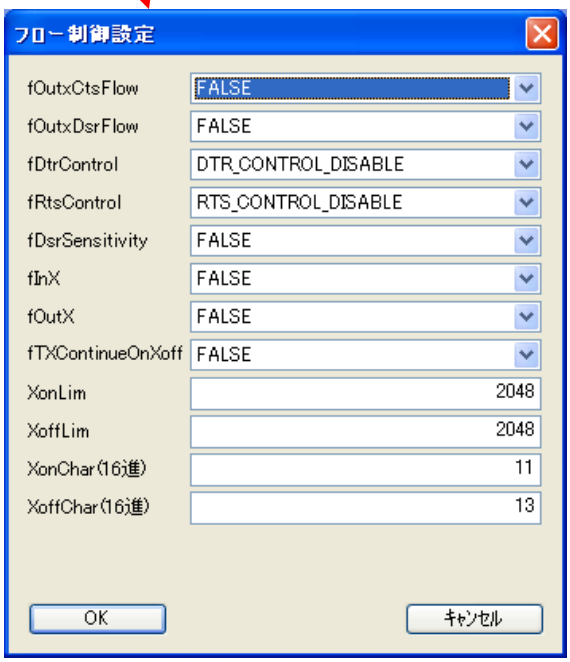
３－５．ＣＯＭ通信（シリアル通信）

COM ポート（シリアル）通信を行う場合の設定について説明します。
左側の項目ペインからポート１（または２）を選択します。
次に「このポートを使用する」にチェックを入れ、通信タイプに「COM」を選択します。









項目解説

通信ポート	シリアル通信で**使用する COM ポート**を設定します。 1～128 まで設定可能です。
ボーレート	**通信速度**を任意に入力します。 右側のコンボボックスから任意の速度を選択することで 左側のテキストボックスに値を挿入することも可能です。
データビット	送信するデータのビット長を設定します。 **8,7,6,5** から選択します。
パリティ	パリティ検出方式を下記から設定します。 none(**NOPARITY**)　　：パリティなし odd(**ODDPARITY**)　　：奇数パリティ even(**EVENPARITY**)　：偶数パリティ mark(**MARKPARITY**)　：マークパリティ space(**SPACEPARITY**)：空白パリティ
ストップビット	ストップビットを下記から設定します。 **1 Bit、1.5 Bit、2 Bit**
フロー制御	通信データのフロー制御を行うか設定します。 **none** の場合はフロー制御を行いません。 **custom** とした場合のフロー制御の条件を細かく設定できます。 右側の設定ボタンからフロー制御設定ダイアログを表示して 条件の入力を行います。設定内容については項目説明（フロー制御設定）で 説明します。
データの区切り	シリアルポートから受信したデータの扱い方法について設定します。 SocketDebugger(SerialDebugger)では、ポートから受信したデータを そのまま受信データとするのでなく、COM 制御ウィンドウの中間バッファに 一時的に表示、蓄積が可能です。蓄積されたデータが受信データとして 扱われる単位について設定が可能です。 **生データ**：受信データはそのまま受信処理へと引き渡されます。 （中間バッファを使用しません。） **固定サイズ**：中間バッファにデータサイズで指定されたバイト数が蓄積 されると受信処理へとデータが渡ります。 例えば３２バイトの設定の場合、はじめに２０バイト受信して、 次にまた２０バイト受信すると先頭から３２バイトが 受信処理へと渡され、残りの８バイトが中間バッファに残ります。 **特定 Char**：特定ｷｬﾗｸﾀが受信データにあった場合、そのｷｬﾗｸﾀまでを含んで、 それまでの蓄積データを受信処理に渡します。キーとなる ｷｬﾗｸﾀ（バイトコード）を「Char 指定」に設定します。（複数可） 例えば、設定が 03(ETX),06(ACK)であった場合、受信したデータに 03 が含まれていると、それまでのデータが受信データとして 処理が行われ、残りは中間バッファに残ります。受信データに 03,06 の両方が含まれていた場合はそれぞれが受信データとなり ２件の受信データ処理が発生することとなります。 ※受信データとして扱わないとは シリアルポートから直接受信したデータを中間バッファに入れるのみで 受信処理として行わないのでメインウィンドウのログに出力されず、 結果、受信後の設定処理なども行われないこととなります。

項目解説（続き）

送信バッファサイズ	シリアルインターフェースで使用する送信バッファのサイズを指定します。 （キーワード：SetupComm 関数）
受信バッファサイズ	シリアルインターフェースで使用する受信バッファのサイズを指定します。 （キーワード：SetupComm 関数）
リトライ周期	通信処理において、エラーが発生した、または終了条件を満たした などで、通信処理が終了した場合に、ここで設定したインターバルを 経過した後、自動で通信処理の実行を行います。
内部プログラム受信バッファサイズ	通信実行時に、内部処理が使用する受信バッファのサイズを設定します。 このサイズにより受信できる最大データ量が決まります。

フロー制御設定

フロー制御設定を custom とした場合のフロー制御の設定を各項目ごとに行います。
（キーワード：SetCommState 関数、DCB 構造体）

fOutxCtsFlow	CTS 出力フロー制御を行うか設定します。 **TRUE**：行う、**FALSE**：行わない
fOutxDsrFlow	DSR 出力フロー制御を行うか設定します。 **TRUE**：行う、**FALSE**：行わない
fDtrControl	DTR フロー制御の種類を設定します。 **DTR_CONTROL_DISABLE** DTR 信号線を無効にします。 **DTR_CONTROL_ENABLE** DTR 信号線を有効にします。 **DTR_CONTROL_HANDSHAKE** ハンドシェイクを有効にします。 手動で DTR 信号線の状態設定を行うとエラーが発生します。
fRtsControl	RTS フロー制御の種類を設定します。 **RTS_CONTROL_DISABLE** RTS 信号線を無効にします。 **RTS_CONTROL_ENABLE** RTS 信号線を有効にします。 **RTS _CONTROL_HANDSHAKE** ハンドシェイクを有効にします。 手動で RTS 信号線の状態設定を行うとエラーが発生します。
fDsrSensitivity	コミュニケーションドライバが DSR 信号線の状態に敏感であるか指定します。このメンバが TRUE であるなら、ドライバーは DSR モデム入力線が高くない場合受信した全てのバイトを無視します。
fInX	受信時に XON/XOFF フロー制御を行うか指定します。 このメンバが TRUE の場合、入力バッファ中の受信データが XoffLim になった場合、XoffChar 文字が送られ、XonLim になった場合 XonChar 文字が送られます。
fOutX	送信時に XON/XOFF フロー制御を行うか指定します。 このメンバが TRUE の場合、XoffChar 文字を受けた場合送信を停止し、XonChar 文字を受けると送信を再開します。
fTXContinueOnXoff	相手の受信バッファが満杯で、ドライバが XoffChar 文字を送った時に送信を停止するか指定します。 このメンバが TRUE の場合、相手の受信バッファがバッファサイズ-XoffLim バイトになり受信停止のために XOFF ｷｬﾗｸﾀを送って来ても送信を続けます。このメンバが FALSE の場合、相手の入力バッファが XonLim になり、XON ｷｬﾗｸﾀを送信再開のために送って来るまで、送信は停止します。
XonLim	XON 文字が送られるまでの受信バッファに許される最小バイト数を 指定します。
XoffLim	XOFF 文字が送られるまでの受信バッファに許される最大バイト数を 指定します。最大許可バイト数は、この値を入力バッファのサイズから 引くことによって計算されます。
XonChar(16 進)	送受信のための XON 文字の値を指定します。
XoffChar(16 進)	送受信のための XOFF 文字の値を指定します。

通信手順

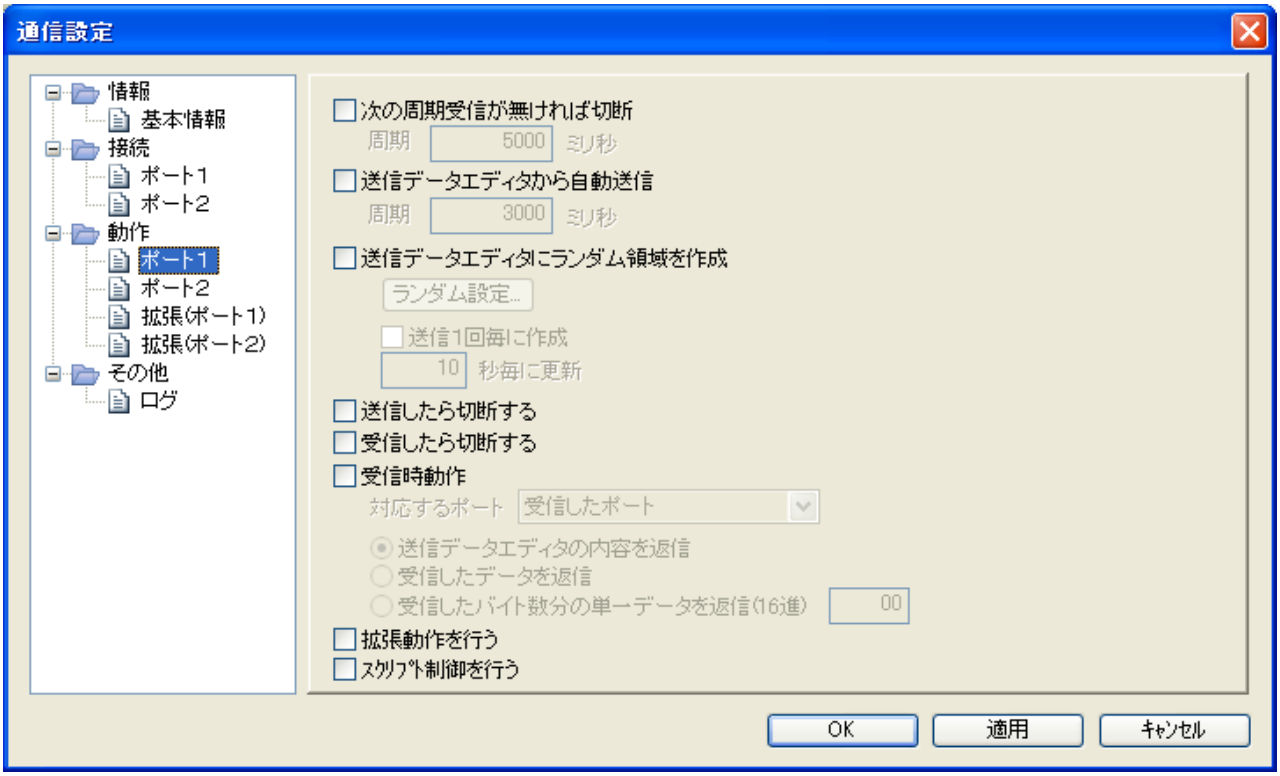
設定後、Port1(または 2)処理開始を選択します。
指定したポートに問題が無ければ処理が開始されます。
この状態で「データ送信」「処理終了」が選択可能となります。

自動処理の設定があれば、設定に従い処理を行いますが、
送信データエディタのデータを手動で「データ送信」や、任意で「処理終了」を行うことが可能です。

COM 制御ウィンドウでは、受信したデータの中間バッファの内容表示、
信号線／エラー状態の確認、信号線状態の設定を行うことが可能です。

３－６．動作設定

通信実行時、データ送受信処理の自動動作定義について説明します。
本設定により、簡易な自動応答処理を行うことが可能です。
左側の項目ペインから動作のうち、ポート１（または２）を選択します。

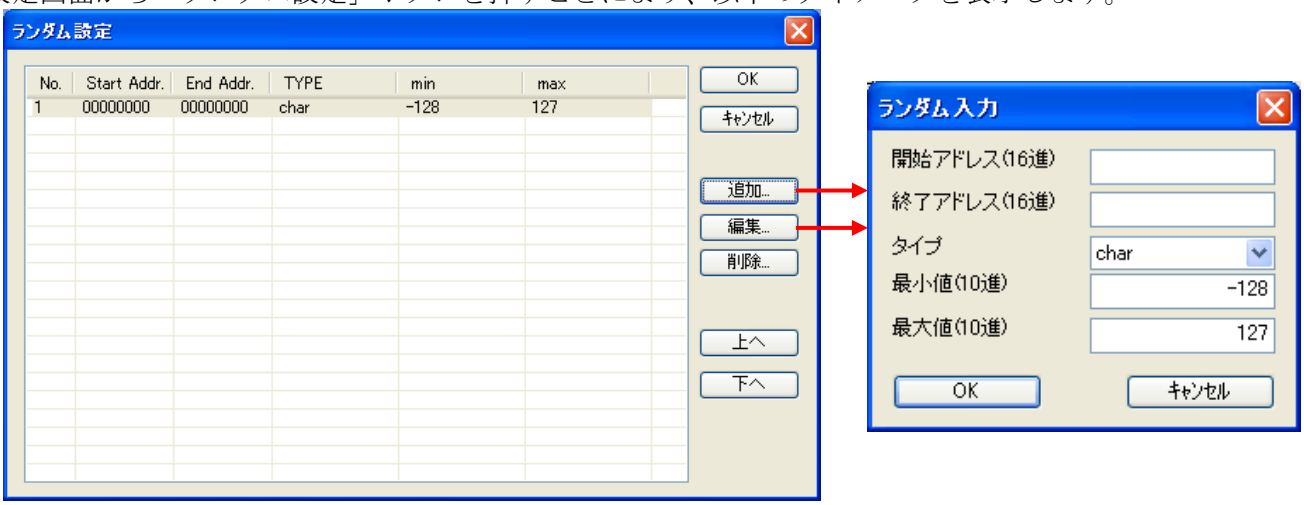


項目解説

次の周期受信が 無ければ切断	有効にした場合、受信タイムアウトの検出を行います。 （周期をミリ秒で設定して下さい。）指定時間内にデータ受信が無かった場合、 回線をクローズし通信処理を終了します。
送信データエディタ から自動送信	有効にした場合、一定周期で送信データエディタの内容を送信します。 （周期をミリ秒で設定して下さい。） データが無い場合は送信は行われません。（ログが出力されます。）
送信データエディタに ランダム領域を作成	送信データエディタから取得される送信データについて、 指定された領域にランダムデータを上書きします。 ランダムデータの更新は 「送信１回毎」と「指定された秒数ごと」の２パターンが設定可能です。 更新されなかった場合、前回のランダムパターンがそのまま利用されます。 ランダムデータの設定方法については次ページで説明します。
送信したら切断する	当該通信ポートでデータの送信を１度でも行った際に、 通信ポートの切断を行います。 ※自動応答処理で送信を行った場合も切断を行いますが、返信するポートが自ポートでない場合、切断条件は実際に送信を行ったポートの設定に依存します。
受信したら切断する	当該通信ポートでデータの受信が１度でも発生した場合に、 通信ポートの切断を行います。
受信時動作	データ受信時に行う自動応答処理を設定します。 チェックが無い場合は、自動応答処理は行いません。 **対応するポート** ○受信したポート データを受信したポートのみが応答します。 （TCP サーバ設定で複数ポート接続していた場合も、 データ受信したポートのみが応答します。） ○ポート１の全接続ポート ポート１の設定で接続している全ポートで応答します。 （TCP サーバ設定で複数ポート接続していた場合、 全ての接続ポートで応答します。） ○ポート２の全接続ポート ポート２の設定で接続している全ポートで応答します。 （TCP サーバ設定で複数ポート接続していた場合、 全ての接続ポートで応答します。） **（対応内容）** ○送信データエディタの内容を返信 送信データエディタに設定されたデータを返信データとして送信します。 ○受信したデータを返信 受信したデータを返信データとして送信します。 ○受信したバイト数分の単一データを返信 受信したバイト数にしたがって、設定されたバイトデータ（１６進）で 埋めて、返信データとして送信します。
拡張動作を行う	応答処理を拡張 DLL で実行します。 本画面で設定した内容は全て無効となります。 詳細は次項の動作設定（拡張）を参照下さい。
ｽｸﾘﾌﾟﾄ制御を行う	応答処理を Lua スクリプトで制御します。 本画面で設定した内容は全て無効となります。 詳細は**６－４．Lua スクリプト拡張説明**を参照下さい。

ランダムデータの設定

ここでは、ランダムデータの入力方法を説明します。
設定画面から「ランダム設定」ボタンを押すことにより、以下のダイアログを表示します。



ランダム設定ダイアログ

設定されたランダム項目一覧を表示します。ランダムデータは本設定に従い生成されます。
No.で指定された順にランダムデータを作成します。設定された領域が重複する場合は、後から生成されたランダムデータで上書きされます。送信データエディタ内のデータが、ランダム設定で指定された領域まで満たない場合、足りない領域分についてはランダムデータの書き込みは行いません。

OK	変更を保存して、ダイアログを閉じます。
キャンセル	変更をキャンセルして、ダイアログを閉じます。
追加	ランダム設定項目を新規追加します。 ランダム入力ダイアログが表示されます。
編集	選択中のランダム設定項目の編集を行います。 ランダム入力ダイアログが既設定データが入力された状態で表示されます。
削除	選択中のランダム設定項目を削除します。 確認ダイアログが表示されます。
上へ	選択中のランダム設定項目の登録順をひとつ上と入れ替えます。
下へ	選択中のランダム設定項目の登録順をひとつ下と入れ替えます。

ランダム入力ダイアログ

個別のランダム設定を入力します。
ランダム上書きエリア（開始アドレス、終了アドレス）の指定、生成されるランダムデータのタイプなどが入力可能です。

開始アドレス(16 進)	上書きエリアの開始アドレスを入力します。
終了アドレス(16 進)	上書きエリアの終了アドレスを入力します。 ここのアドレスまでが上書き対象となります。設定されたタイプごとのランダムデータを生成しつつ埋めていきます。 ※領域が足りない場合ランダムデータは領域のサイズで切られます。
タイプ	生成されるランダムデータのタイプを設定します。 次ページの一覧を参照下さい。
最小値	ランダム生成される値の範囲（最小値）を設定します。
最大値	ランダム生成される値の範囲（最大値）を設定します。
OK	設定を登録して、ダイアログを閉じます。
ｷｬﾝｾﾙ	設定をキャンセルして、ダイアログを閉じます。

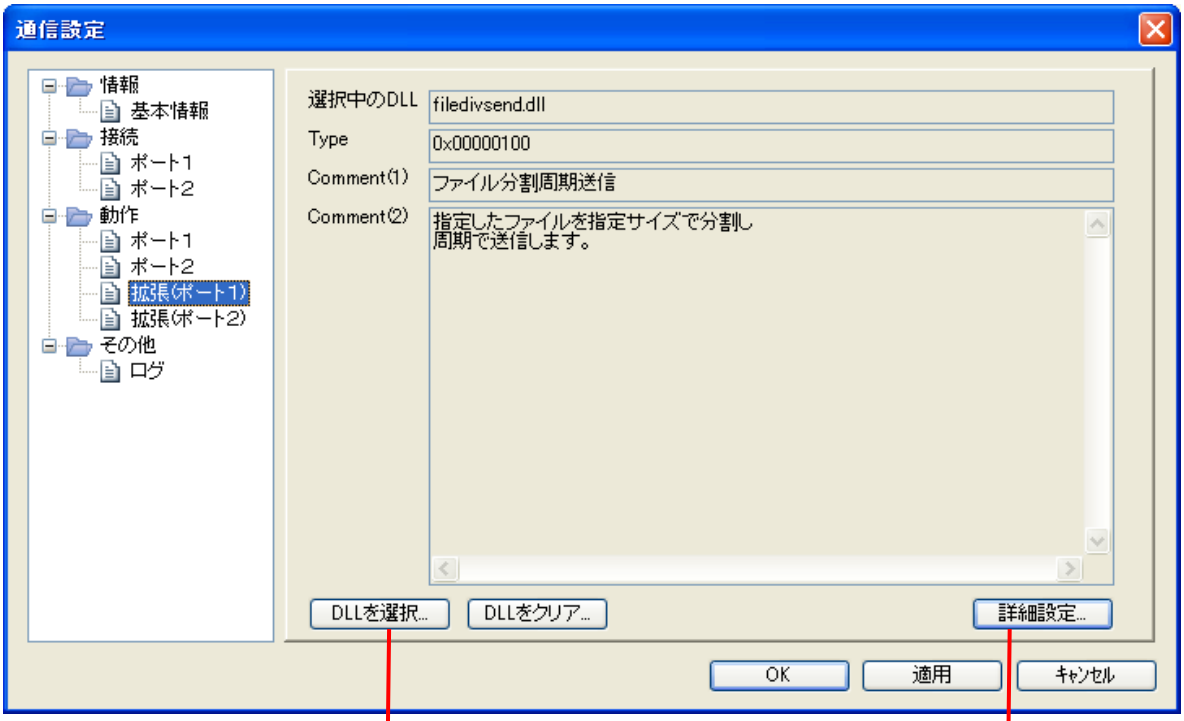
ランダム入力ダイアログ（タイプ一覧）

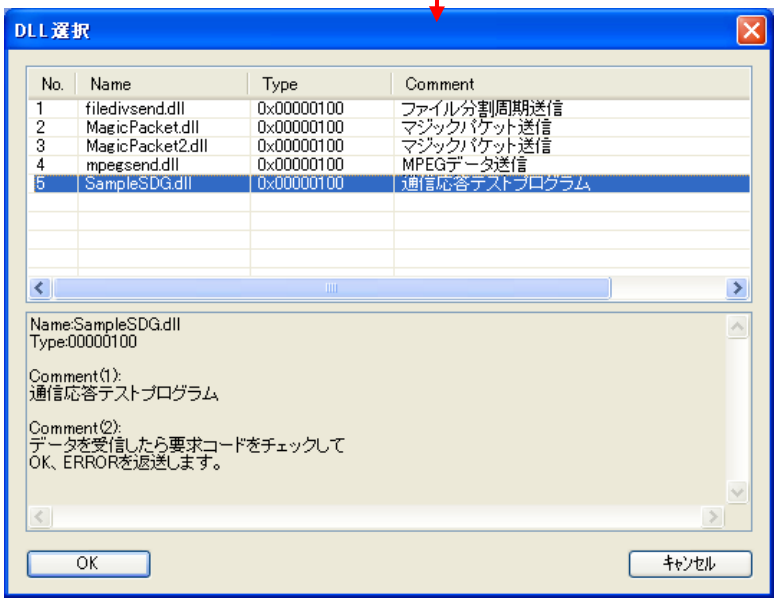
タイプ名の後ろに（ＬＥ）（ＢＥ）と付いた項目がありますが、
（ＬＥ）はリトルエンディアン
（ＢＥ）はビッグエンディアン
の意味です。条件にしたがって、バイト配列を並べ替えます。
各タイプごとに生成する数値範囲の設定が可能です。

タイプ	最小値	最大値	説明
char	-128	127	1 バイトの符号付き整数を生成します。
BYTE	0	255	1 バイトの符号無し整数を生成します。
short	-32768	32767	2 バイトの符号付き整数を生成します。
WORD	0	65535	2 バイトの符号無し整数を生成します。
long	-2147483648	2147483647	4 バイトの符号付き整数を生成します。
ULONG	0	4294967295	4 バイトの符号無し整数を生成します。
float	float 最小値	float 最大値	4 バイト指数による浮動小数点データを生成します。
double	double 最小値	double 最大値	8 バイト指数による浮動小数点データを生成します。

３－７．動作拡張設定（拡張 DLL）

通信実行時の自動データ送受信処理として専用に作成・提供された DLL ファイルを選択し、
ユーザープログラミングによる自動応答処理を実行することが可能です。
本設定画面から DLL を選択し、必要であれば DLL で提供される詳細設定ダイアログから要求される
通信情報の設定を行います。
※拡張 DLL ファイルは SocketDebugger 本体が稼動しているフォルダに配置しておく必要があります。

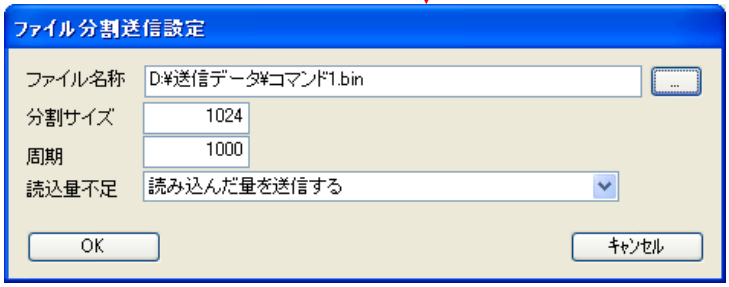




使用する DLL を選択します。

画面上部に選択可能な DLL の一覧が表示されます。
画面下部には表中で選択されている DLL の概要情報が表示されます。

下記ダイアログは一例です。
選択された DLL により
ダイアログは異なります。

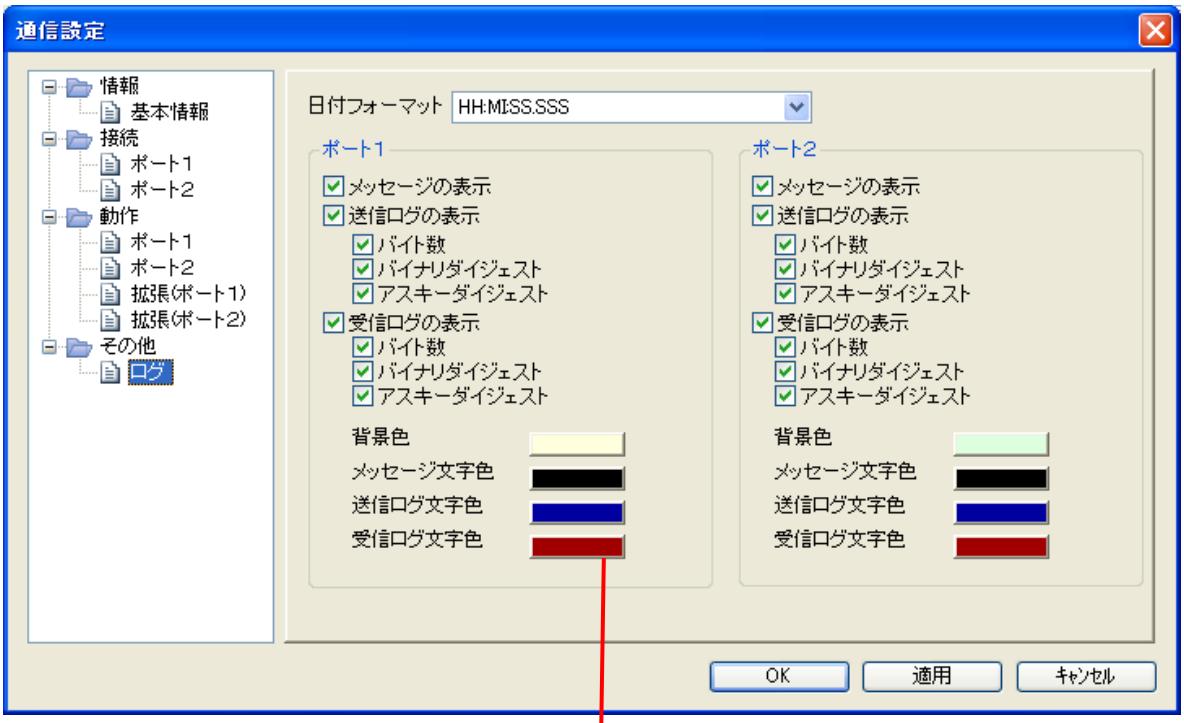


項目解説

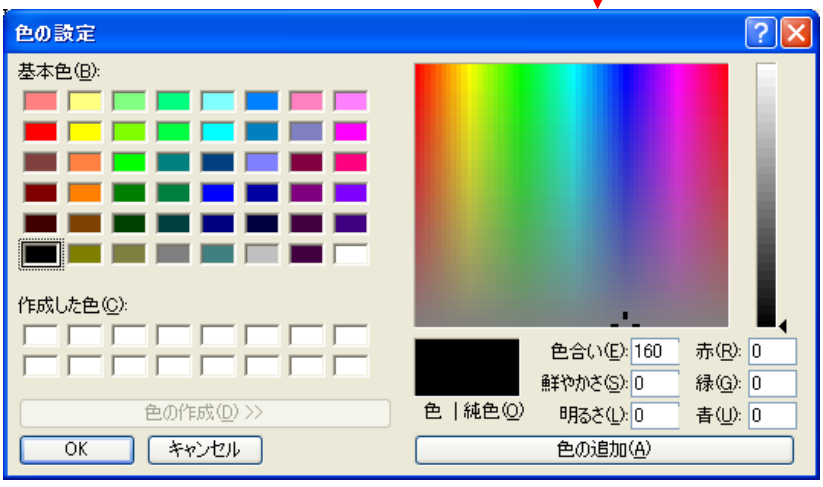
選択中の DLL	現在設定されている DLL ファイルの名称が表示されます。
Type	DLL のタイプが表示されます。 将来的に作成できる DLL のタイプが追加された場合の拡張用です。 現在は 0x100 のみです。
Comment(1)	DLL の処理概要（タイトル）が表示されます。
Comment(2)	DLL の処理概要（説明）が表示されます。
「DLL を選択」ボタン	DLL 選択ダイアログが表示されます。 SocketDebugger が動作しているフォルダ内に配置されている DLL が 上部表に列挙されますので、任意の DLL の選択を行って下さい。
「DLL をｸﾘｱ」ボタン	現在設定されている DLL ファイルをクリアします。 確認ダイアログが表示されます。
「詳細設定」ボタン	DLL に動作設定が可能な場合、ボタンが押せるようになります。 各 DLL が用意した設定ダイアログが表示されますので 必要な情報を入力して下さい。 ここで設定した情報についても、通信設定ファイルに保存が行われます。

3－8．ログ

メインウィンドウに表示されるログ情報の表示フィルタ、色の指定を行います。
表示されるログのうち、特定の項目のみ表示したい場合に設定を調整します。変更後は、現在表示中のログについても設定に従って再表示を行います。
左側の項目ペインからその他、ログを選択します。

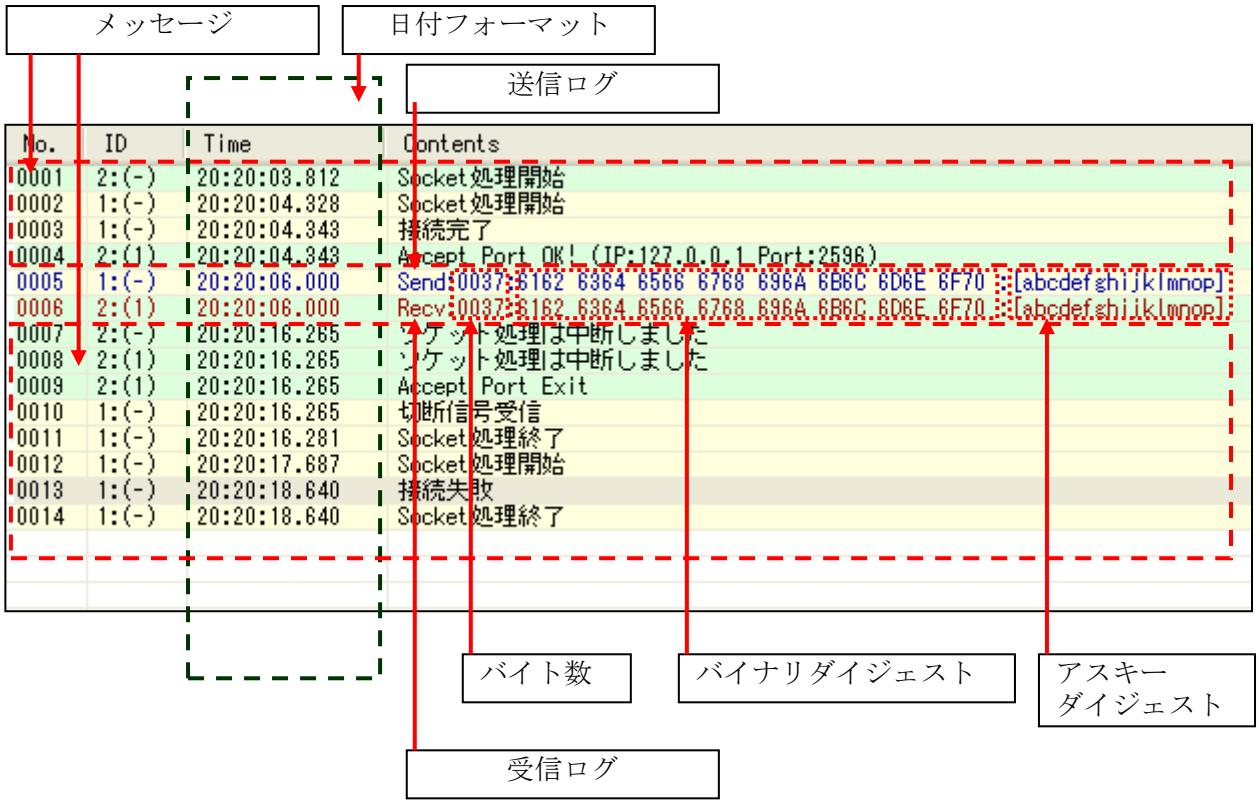


各、色ボタンを押すことにより、
色の設定ダイアログを表示します。



項目解説

日付フォーマット	表示される日付フォーマットを指定します。 HH:MI:SS.SSS　　：時、分、秒、ミリ秒 YYYY/MM/DD HH:MI:SS.SSS　：年（４桁）、月、日、時、分、秒、ミリ秒
下記からはポート１、ポート２でそれぞれに設定が可能です。	
メッセージの表示	通信の開始、終了、エラー通知などの文字メッセージの出力の ON/OFF を設定します。
送信ログの表示	当該ポートの送信ログの出力を ON/OFF します。 また、バイト数、バイナリダイジェスト、アスキーダイジェスト についても個別に ON/OFF の設定が可能です。
受信ログの表示	当該ポートの受信ログの出力を ON/OFF します。 また、バイト数、バイナリダイジェスト、アスキーダイジェスト についても個別に ON/OFF の設定が可能です。
背景色	当該ポートログの背景色を設定します。 色の選択ダイアログを表示します。
メッセージ文字色	当該ポートログのメッセージ文字色を設定します。 色の選択ダイアログを表示します。
送信ログ文字色	当該ポートログの送信ログ文字色を設定します。 色の選択ダイアログを表示します。
受信ログ文字色	当該ポートログの受信ログ文字色を設定します。 色の選択ダイアログを表示します。



３－９．ソケットオプションの設定

ソケットを扱う通信の場合、ソケットオプションの設定が可能です。
行う通信の種別により、意味をなさない設定についても、設定通りにソケットを設定します。
（したがって、設定を行っても正しく動作するかは実装に依存することとなります。）
設定を行いたい項目にチェックを入れて、パラメータを設定してください。

SOL_SOCKET	
SO_REUSEADDR	ポート再利用の設定
SO_KEEPALIVE	生存確認信号の設定
IPPROTO_IP	
IP_TTL	Time To Live の設定。
IP_MULTICAST_TTL	マルチキャスト時の TTL の設定
IP_MULTICAST_LOOP	マルチキャストの自己ループの設定

※各パラメータの詳細は Winsock の情報などを参照して下さい。
※上記のほか、
SOL_SOCKET, SO_RCVBUF
SOL_SOCKET, SO_BROADCAST
IPPROTO_IP, IP_ADD_MEMBERSHIP
IPPROTO_IP, IP_DROP_MEMBERSHIP
が、通信設定により使用されています。

設定と動作の反映

	TCP Client	TCP Server	UDP UNICAST	UDP BROADCAST	UDP MULTICAST
SO_REUSEADDR	△	△	○	○	○
SO_KEEPALIVE	○	○	×	×	×
IP_TTL	△	△	○	△	△
IP_MULTICAST_TTL	×	×	△	△	○
IP_MULTICAST_LOOP	×	×	△	○	○

○	設定可能
△	動作せず
×	ERROR

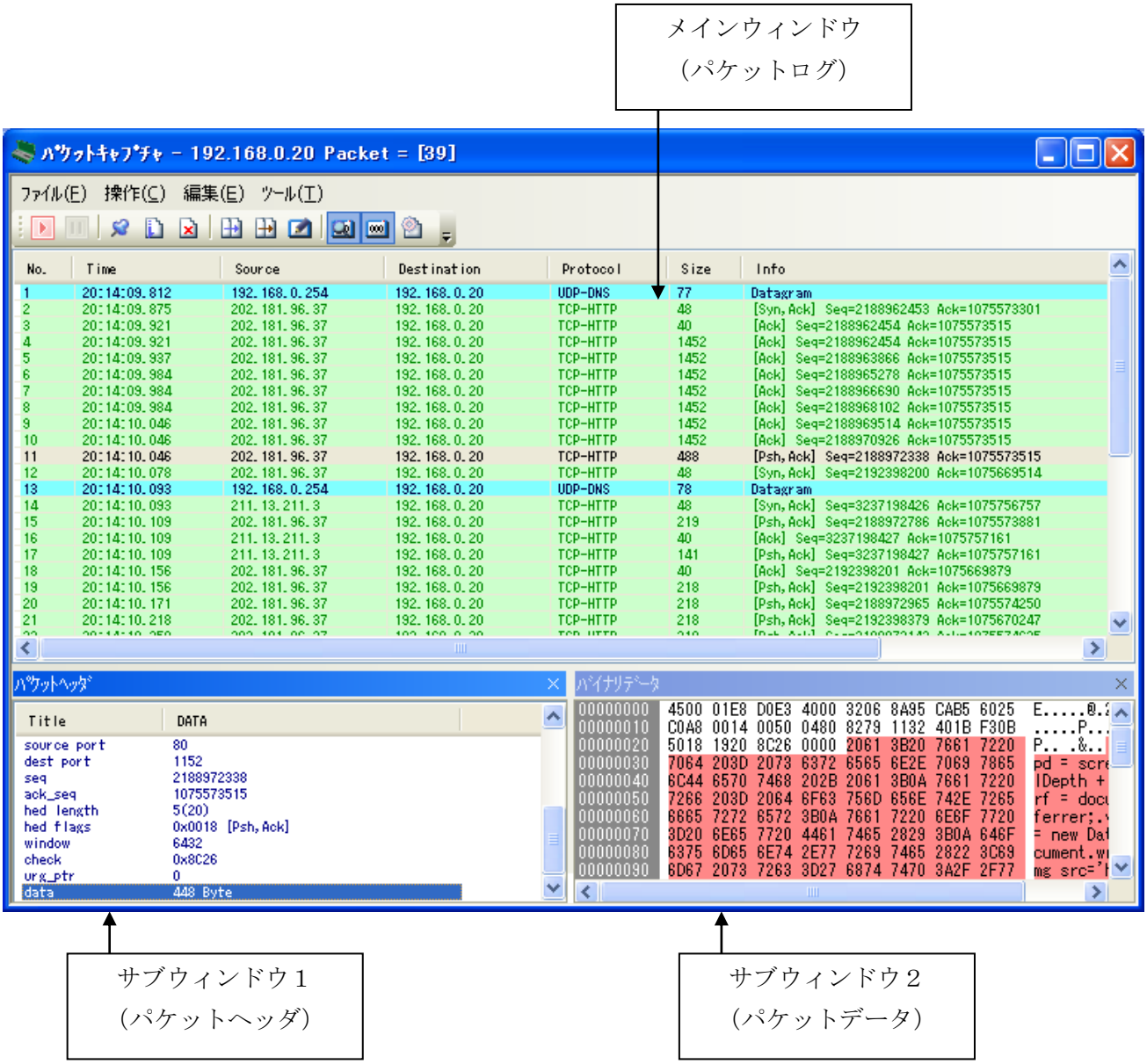
※動作対応を調べた結果が上記ですが、UDP-UNICAST の IP_TTL で値を 32 に設定すると動作が 128 になるなど、不条理な状況が確認できました。詳細を調べた方がいましたらご教示下さい。

パケットキャプチャ

SocketDebugger では、ネットワーク上に流れるパケットのキャプチャと、キャプチャデータの再利用が可能です。
ここではパケットキャプチャ機能の使い方を説明します。

４－１．機能概略

本機能は SocketDebugger とは独立したウィンドウで表示されます。
呼び出すには SocketDebugger のメニューから「ﾂｰﾙ」－「ﾊﾟｹｯﾄｷｬﾌﾟﾁｬ」を選択して下さい。
初期ウィンドウ表示時は以下のように表示されます。
ここでは既にサブウィンドウ（２枚）が張り付いた状態となっています。
キャプチャしたパケットはメインウィンドウに時系列で表示されます。
任意のログを選択することで、選択されたパケットデータについて
パケットヘッダ情報とパケット（バイナリ）データがそれぞれのサブウィンドウに表示されます。
（ヘッダ表示で対応しているプロトコルは TCP, UDP, ICMP, IGMP です。）
キャプチャしたデータは、データ部、全データ、分割組み立ての３パターンで
そのまま送信データエディタにセットし、テスト用送信データとして再利用することも可能です。
また、パケットログの保存、読み込みを行うことができ、Ethereal で保存されたパケットログの
読み込みにも対応しています。



注意

１．パケットキャプチャ機能は、Winsock2 の RAW ソケット機能により実現しています。
　　そのため、IP 層より下の階層のパケットはキャプチャを行うことができません。
２．実行するためには PC のユーザ権限において、管理者権限が必要となります。
３．一部の NIC、ドライバの仕様により自端末が出力したパケットのキャプチャを行えない
　　場合があります。
４．実行させる端末のスペックによりパケットデータの取りこぼし等が発生する可能性があります。

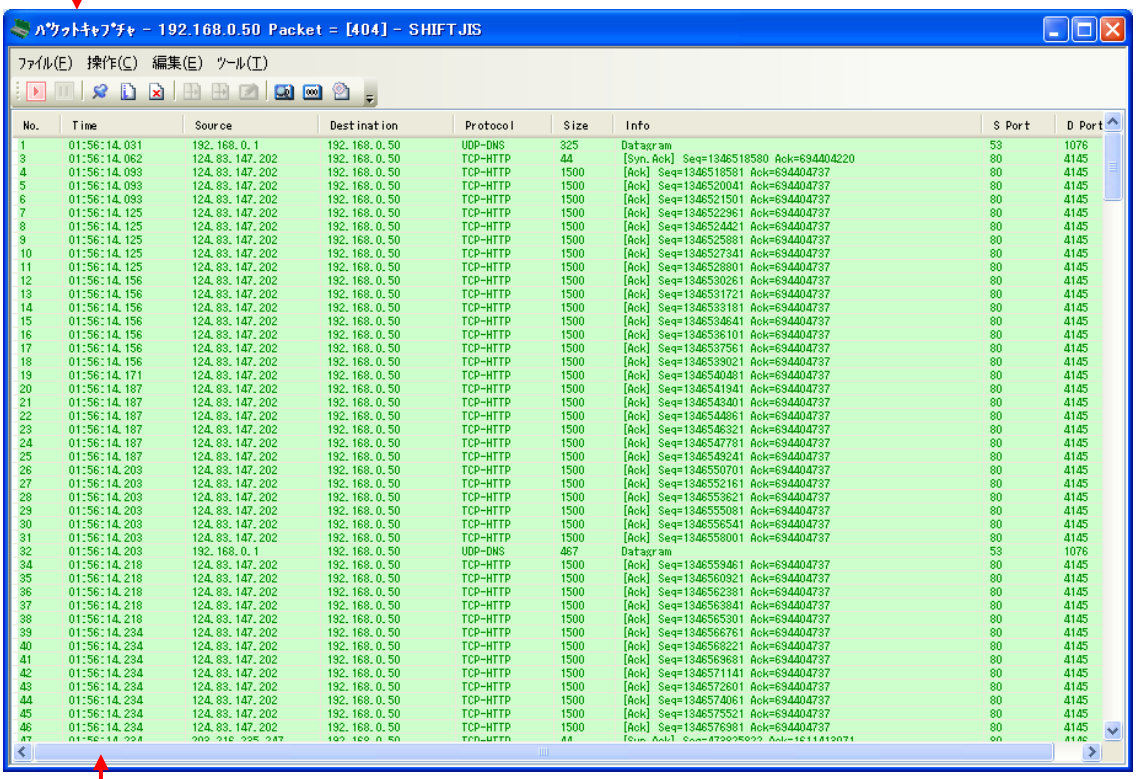
Etherealとは

フリーの LAN アナライザソフトです。GTK+ を利用して開発されており
使用できるプラットフォーム（OS）が多彩で Windows、Mac、UNIX、Linux 等で利用できます。
Windows の場合、インストールには WinPcap というフリーのパケットキャプチャドライバが必要です。
Ethereal も WinPcap もインストーラがあるため簡単にインストールが可能です。
Ethreal は MAC 層からのキャプチャを行うことができます。

４－２．メインウィンドウ

パケットキャプチャのメインとなるウィンドウについて説明します。
表示内容はキャプチャしたパケットログの一覧です。ログの表示内容については次ページで説明します。
本ウィンドウは、メニュー、ツールバー、を含んでおり、パケットキャプチャについて総合的な
操作を行います

タイトル：キャプチャを行っている NIC の IP アドレス、
現在取得したパケットの数、表示文字コード
を表示します。



パケットログ：取得したパケットの基本的な情報を表示します。

項目解説

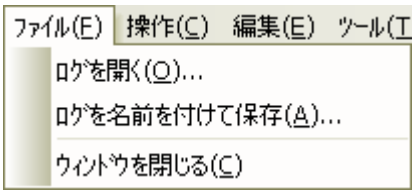
No.	管理しているパケットログの取得番号を表示します。
Time	パケットログを取得した時刻を表示します。
Source	パケットの送出元 IP アドレスを表示します。
Destination	パケットの送出先（宛て先）IP アドレスを表示します。
Protocol	キャプチャしたパケットのプロトコルを表示します。
Size	取得したパケットログのデータサイズを表示します。 基本的にはパケットヘッダ＋プロトコルヘッダ＋データのサイズとなります。
Info	プロトコルごとにパケット情報についてヘッダ情報の一部などを 表示します。
S Port	パケットの送出元ポート番号を表示します。
D Port	パケットの送出先（宛て先）ポート番号を表示します。

表示されている色については、フィルタの設定により着色されます。

詳細は「４－６．表示フィルタ」を参照して下さい。

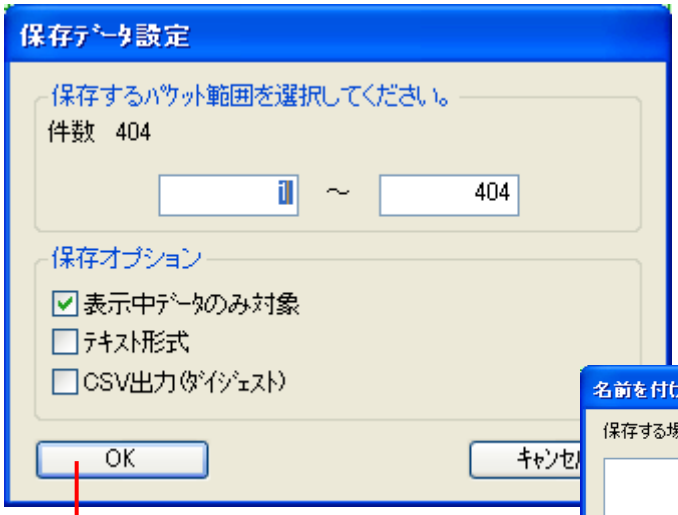
メニュー解説

「ファイル」



ﾛｸﾞを開く	
保存されているパケットログファイルを読み込みます。 SocetDebugger で保存されたログファイルの他、 Ethereal で保存されたログファイルについても読み込み可能です。	
ﾛｸﾞを名前を付けて保存	
メインウィンドウで表示しているパケットログに名前を付けて ファイルに保存します。選択すると、パケット番号、保存形式選択ダイアログが表示されます。	
ｳｨﾝﾄﾞｳを閉じる	
パケットキャプチャ機能を終了します。 パケットログは削除されますのでご注意下さい。	

パケット番号、保存形式選択ダイアログ



表示中ﾃﾞｰﾀのみ対象

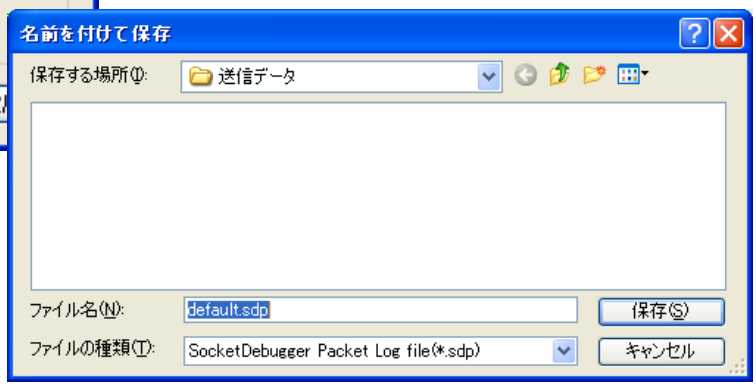
表示ﾌｨﾙﾀに従ったﾊﾟｹｯﾄのみが保存されます。

ﾃｷｽﾄ形式

ﾛｸﾞをﾃｷｽﾄﾌｧｲﾙに出力します。

CSV 出力

ﾛｸﾞのﾀﾞｲｼﾞｪｽﾄを CSV ﾌｧｲﾙに出力します。



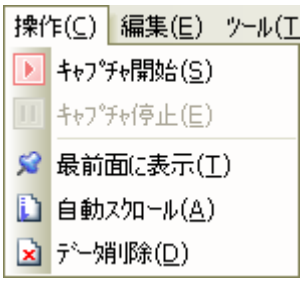
※Ethereal からのファイル読み込みについて

SocketDebugger が読み込むのは IP 層からのデータとなります。他のデータは読み飛ばされます。

（ARP 等の情報も欠落します。）

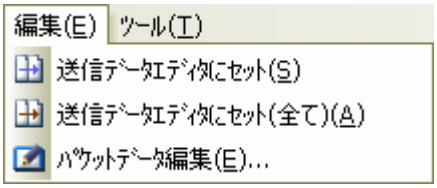
また、MAC 層の情報、ミリ秒以下のタイムスタンプ等についても読み飛ばします。

「操作」



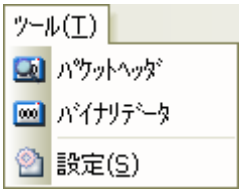
ｷｬﾌﾟﾁｬ開始	
キャプチャを開始します。既にキャプチャログがある場合、 取得したキャプチャログは追加されていきます。	
ｷｬﾌﾟﾁｬ停止	
キャプチャ処理を停止します。	
最前面に表示	
メインウィンドウの最前面表示切替を行います。最前面に表示中はチェックアイコンが表示されます。チェックを外せば元の状態に戻ります。	
自動ｽｸﾛｰﾙ	
ログにデータが追加された際に自動で、「最新データを選択」「可視位置までのスクロール」を行うか指定します。指定中はチェックアイコンが表示されます。チェックを外せば元の状態に戻ります。	
ﾃﾞｰﾀ削除	
現在取得しているメインウィンドウのパケットログ情報を削除します。 確認ダイアログが表示されます。	

「編集」



送信ﾃﾞｰﾀｴﾃﾞｨﾀにｾｯﾄ	
現在選択されているパケットログのデータ部のバイトデータを 送信データエディタにコピーします。 キャプチャしたデータの再利用を行う場合に使用します。	
送信ﾃﾞｰﾀｴﾃﾞｨﾀにｾｯﾄ(全て)	
現在選択されているパケットログの全てのバイトデータを 送信データエディタにコピーします。 IP ヘッダについてもコピーされるため、パケットの再現という 意味では使用できません。	
ﾊﾟｹｯﾄﾃﾞｰﾀ編集	
現在選択されているパケットの情報を同一通信データについて 選択パケット以降で検索し、データの連結用ダイアログ 「パケット編集」に表示します。 詳細は「４－８．パケット編集」を参照して下さい。	

「ﾂｰﾙ」

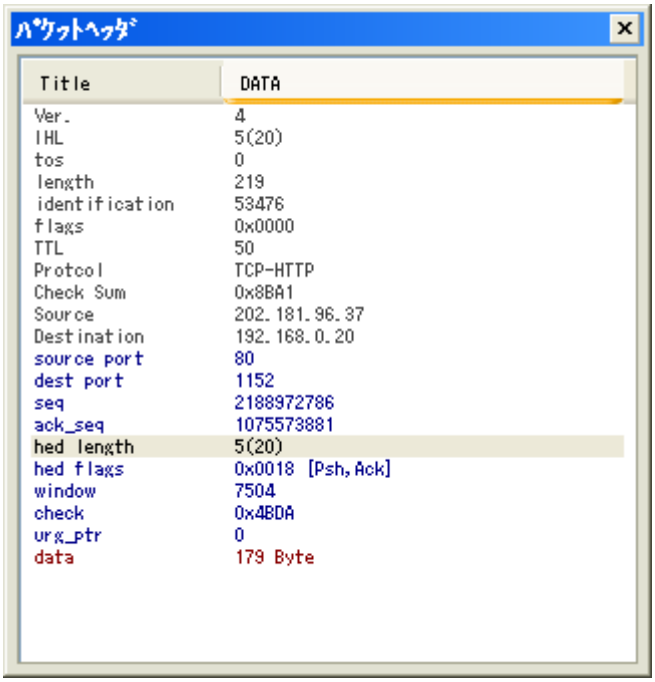


ﾊﾟｹｯﾄﾍｯﾀﾞ	
(ｻﾌﾞｳｨﾝﾄﾞｳ)ﾊﾟｹｯﾄﾍｯﾀﾞの表示/非表示を切り替えます。	
ﾊﾞｲﾅﾘﾃﾞｰﾀ	
(ｻﾌﾞｳｨﾝﾄﾞｳ)ﾊﾞｲﾅﾘﾃﾞｰﾀの表示/非表示を切り替えます。	
設定	
キャプチャ設定ダイアログを表示します。 詳細は 「４－５．キャプチャ設定」 「４－６．表示フィルタ設定」 「４－７．色フィルタ設定」を参照して下さい。	

４－３．パケットヘッダ

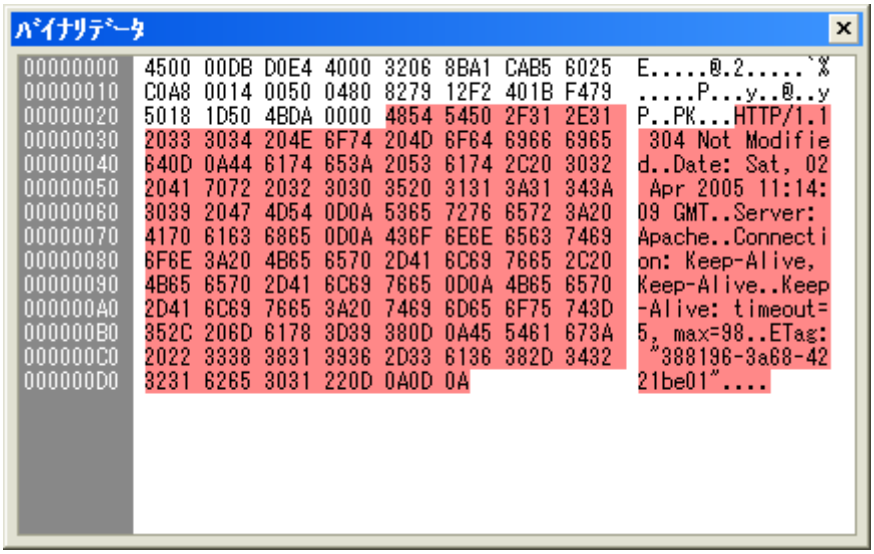
メインウィンドウで選択したパケットデータのヘッダ内容を表示します。
グレーの部分が IP ヘッダ、青の部分がプロトコル別のヘッダ、赤の部分がデータ領域で区別されます。
本ウィンドウの項目を選択することにより、バイナリデータウィンドウで該当するデータ箇所が
赤く表示されます。
認識できるプロトコルは下記となります。

- TCP
- UDP
- ICMP
- IGMP



４－４．バイナリデータ

メインウィンドウで選択したパケットデータの内容をバイナリデータで表示します。
ヘッダ、データ部ともに表示を行います。
パケットヘッダで選択された項目の領域について背景を赤色で表示します。



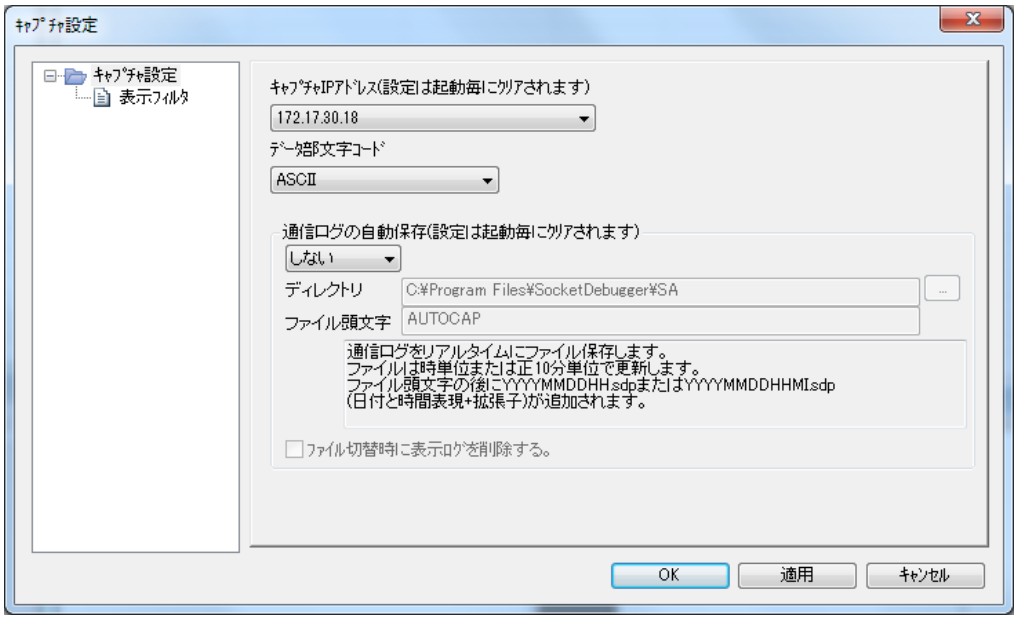
４－５．キャプチャ設定

キャプチャを行うための基本設定を行います。
左側の項目ペインからキャプチャ設定を選択します。
キャプチャを行う対象の IP アドレスを指定します。
（ここで指定された IP を保持する NIC を選択したこととなります。複数のアドレスを持つ NIC であれば
その NIC が受け取るパケットが対象となります。）

データ部文字コードは、任意の漢字コードを指定することで
バイナリエディタ表示のテキスト部を漢字変換し表示を行います。

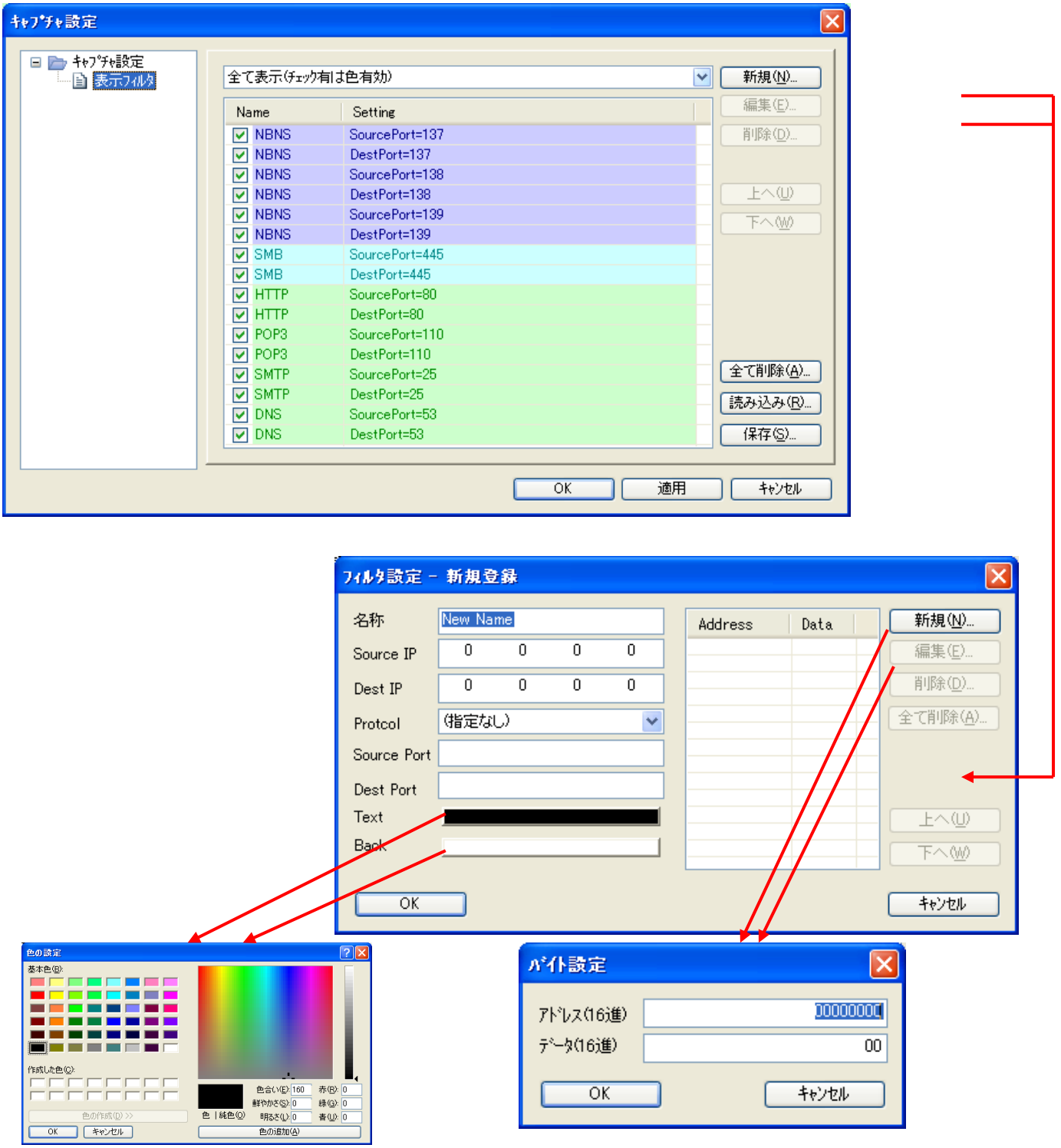
指定した IP アドレス、データ部文字コードはメインウィンドウのタイトルに表示されます。

通信ログの自動保存を選択する（時単位または１０分単位）と、キャプチャしたパケットを自動でファイルに保存します。ファイルは１時間（または正１０分単位）ごとに別名に切り替わります。
ファイルを保存するフォルダと、ファイル名の先頭を入力して下さい。
この機能は、長時間パケットをキャプチャして後に解析を行う場合などに使用します。



４－６．表示フィルタ設定

取得した全てのパケットをメインウィンドウに表示すると、関係のないパケットが多い場合、
非常に見づらい表示となります。ここでは、メインウィンドウに表示を行うパケットの
フィルタ（選択）、表示色指定について設定を行います。
左側の項目ペインから表示フィルタを選択します。



フィルタは複数登録することが可能です。また、登録してあるフィルタ群について一括で
無効、表示する、表示しないを選択できます。フィルタ項目ごとにチェックマークで対象として「扱う」、
「扱わない」の指定が可能です。
（例えば、
「ﾁｪｯｸ有を表示する」でチェック有の項目は表示対象です。
「ﾁｪｯｸ有を表示しない」でチェック有の項目は表示対象外です。
「全て表示する(ﾁｪｯｸ有は色有効)」の場合は全てのパケットが表示されますが、
ﾁｪｯｸの付いたもののみ、色が反映されます。）
表示しないパケットについても、データの保持は行っているため、キャプチャ後にフィルタ設定を変更
することで表示を行うようにすることも可能です。

フィルタは登録順に（上から）判定され、パケットがフィルタに該当した場合に指示された表示方法を
行い、以後のチェックは行いません。

登録したフィルタはファイルに保存して、読み込むことが可能ですので、作業にあわせたフィルタパターンを
簡単に切り替えることができます。
保存を行わなくても登録した情報はパケットキャプチャウィンドウの終了時に自動で
ファイル「default.sdc」に保存されるため、以降は自動で最後の設定を読み込みます。

デフォルトで代表的ないくつかの設定が登録されています。
デフォルトに戻したい場合はファイル「default.sdc」を削除して再起動してください。

項目解説（キャプチャ設定－表示フィルタ）

（処理指定）	「全て表示する(ﾁｪｯｸ有は色有効)」「ﾁｪｯｸ有を表示する」「ﾁｪｯｸ有を表示しない」の３つから処理方法を指定します。 全て表示する：全てのパケットが表示対象。 ﾁｪｯｸ有を表示する：フィルタに該当したパケットのみ表示 ﾁｪｯｸ有を表示しない：フィルタに該当したパケットは非表示。それ以外は表示。
（表示フィルタ一覧）	登録されているフィルタの一覧を表示します。 上から優先順位が高いフィルタとなります。 チェックマークを外すことで登録フィルタを個別に無効とすることができます。
新規	フィルタ群の最後に新しく項目を追加します。 新規登録ダイアログが表示されます。
編集	既に登録されているフィルタの編集（変更）を行います。 編集ダイアログが表示されます。
削除	一覧で選択しているフィルタを削除します。 確認ダイアログが表示されます。
上へ	一覧で選択しているフィルタの優先順位（表示位置）をひとつ上と入れ替えます。
下へ	一覧で選択しているフィルタの優先順位（表示位置）をひとつ下と入れ替えます。
全て削除	登録しているフィルタを全て削除します。 確認ダイアログが表示されます。
読み込み	ファイルに保存されている表示フィルタファイルを選択して読み込みます。 デフォルトの拡張子は. sdc です。 現在登録されているフィルタは削除されますのでご注意下さい。
保存	現在のフィルタ設定に名前をつけて保存します。 （保存される情報は各フィルタとチェック情報です。 処理指定は保存されません）

項目解説（フィルタ設定－新規登録（編集））

名称	一覧に表示されるフィルタの名称を入力します。
SourceIP	パケットの送信元 IP アドレスを指定します。(0.0.0.0 で指定なし)
DestIP	パケットの送信先 IP アドレスを指定します。(0.0.0.0 で指定なし)
Protcol	プロトコルを下記から指定します。 （指定なし)、TCP、UDP、ICMP、IGMP
SourcePort	パケットの送信元ポート番号を指定します。(空白で指定なし)
DestPort	パケットの送信先ポート番号を指定します。(空白で指定なし)
Text	メインウィンドウに表示するパケットデータの文字色を選択します。 色選択ダイアログが表示されます。
Back	メインウィンドウに表示するパケットデータの背景色を選択します。 色選択ダイアログが表示されます。
新規登録	バイト判定データ群の最後に新しくバイト判定データを追加します。 バイト設定ダイアログが表示されます。
編集	既に登録されているバイト判定データの編集（変更）を行います。 バイト設定ダイアログが表示されます。
削除	バイト判定一覧で選択しているバイト判定データを削除します。 確認ダイアログが表示されます。
全て削除	バイト判定一覧で選択しているバイト判定データを全て削除します。 確認ダイアログが表示されます。
上へ	一覧で選択しているバイト判定データの表示位置をひとつ上と入れ替えます。 優先順位の概念がないため、順番入れ替えが意味するものは 表示上の見易さのみです。
下へ	一覧で選択しているバイト判定データの表示位置をひとつ下と入れ替えます。 優先順位の概念がないため、順番入れ替えが意味するものは 表示上の見易さのみです。
OK	入力したデータの追加（または変更の反映）を行い ダイアログを終了します。 入力データに誤りがある場合は通知ダイアログが表示されます。
キャンセル	入力データを破棄してダイアログを終了します。

※注意１
登録されている各項目は AND 判定で処理を行います。
例えば、送信ポート番号８０、受信ポート番号が８０の場合、両方が８０の通信のみが対象となります。
８０ポートの両方向への通信をキャプチャしたい場合は、送信ポート８０の設定と受信ポート８０の設定で
フィルタ登録が２つ必要となります。

※注意２
バイト判定で通信データの判定を行う場合、ヘッダに付加情報がついた場合、参照アドレスが
指定したいアドレスとずれる場合はありますのでご注意下さい。

項目解説（バイト設定ダイアログ）

アドレス	キャプチャデータの先頭を０とした、参照アドレスを指定します。 （入力は１６進数） キャプチャデータが、指定アドレスに満たない場合は、判定不能として 不一致扱いとなります。
データ	判定するデータを１バイト指定します。（入力は１６進数） 複数のバイトを指定したい場合は、個別に登録を行って下さい。
OK	入力したデータの追加（または変更の反映）を行い ダイアログを終了します。
キャンセル	入力データを破棄してダイアログを終了します。

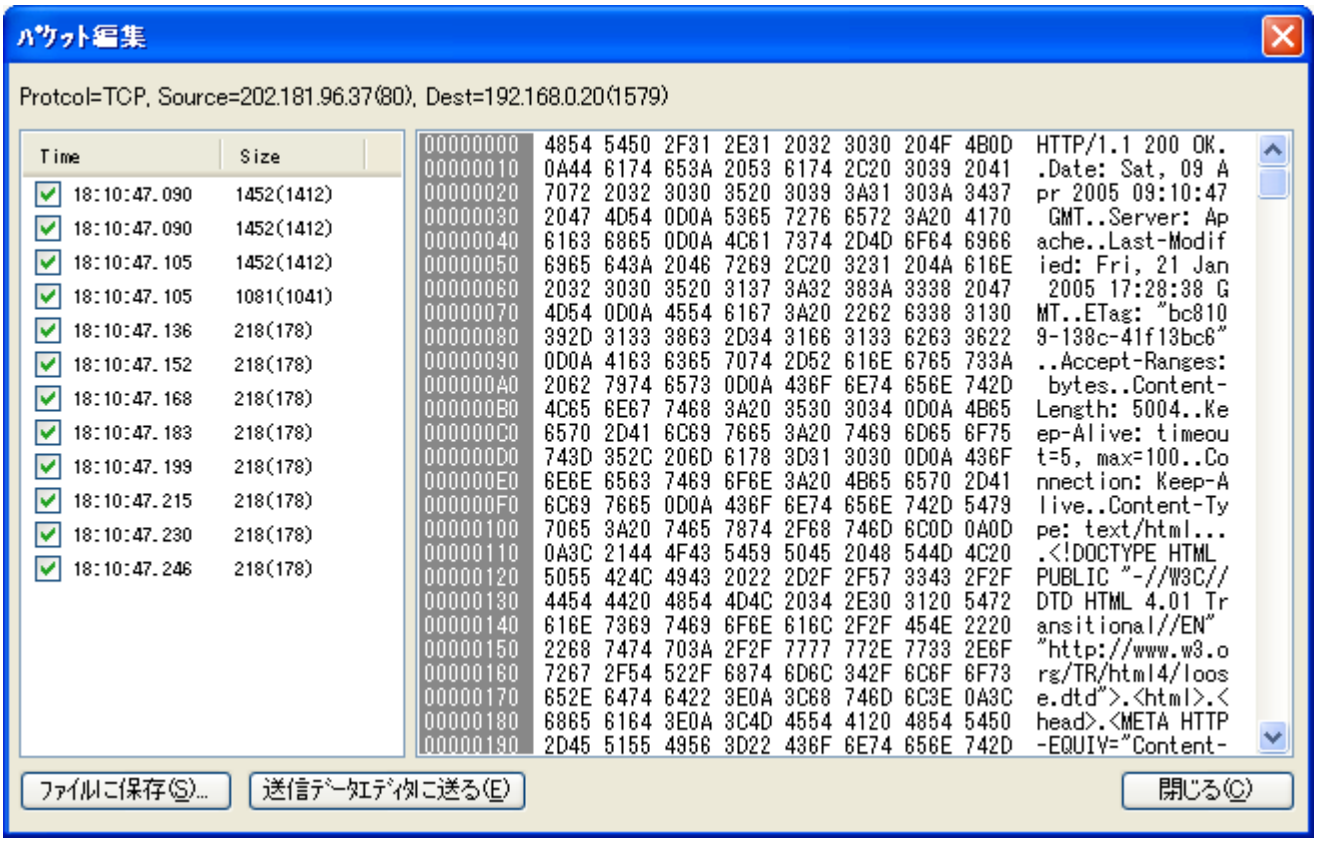
４－７．パケット編集

キャプチャで取得したパケットデータについて、大きいデータは分割して送受信が行われていることが確認できます。SocketDebugger では、分割されたパケットデータを結合し、
データの再利用（送信データとしての利用）を行うことが可能です。
ここでは、パケット編集の手順と制約について説明します。

下記に示す、例はブラウザで WEB ページにアクセスを行った際のキャプチャ例です。
パケットデータが大きいため、分割されています。
結合を行いたいパケットの先頭を選択して、メニューから「編集」－「パケットデータ編集」を選択します。

1	18:10:47.074	202.181.96.37	192.168.0.20	TCP-HTTP	48	[Syn,Ack] Seq=1489978955 Ack=2950692050	80	1579
2	18:10:47.090	202.181.96.37	192.168.0.20	TCP-HTTP	40	[Ack] Seq=1489978956 Ack=2950692493	80	1579
3	18:10:47.090	202.181.96.37	192.168.0.20	TCP-HTTP	1452	[Ack] Seq=1489978956 Ack=2950692493	80	1579
4	18:10:47.090	202.181.96.37	192.168.0.20	TCP-HTTP	1452	[Ack] Seq=1489980368 Ack=2950692493	80	1579
5	18:10:47.105	202.181.96.37	192.168.0.20	TCP-HTTP	1452	[Ack] Seq=1489981780 Ack=2950692493	80	1579
6	18:10:47.105	202.181.96.37	192.168.0.20	TCP-HTTP	1081	[Psh,Ack] Seq=1489983192 Ack=2950692493	80	1579
7	18:10:47.136	202.181.96.37	192.168.0.20	TCP-HTTP	48	[Syn,Ack] Seq=1500828307 Ack=2950737915	80	1580
8	18:10:47.136	202.181.96.37	192.168.0.20	TCP-HTTP	218	[Psh,Ack] Seq=1489984233 Ack=2950692866	80	1579
9	18:10:47.152	211.13.211.3	192.168.0.20	TCP-HTTP	141	[Psh,Ack] Seq=3356132871 Ack=2939250253	80	1577
10	18:10:47.152	202.181.96.37	192.168.0.20	TCP-HTTP	40	[Ack] Seq=1500828308 Ack=2950738289	80	1580
11	18:10:47.152	202.181.96.37	192.168.0.20	TCP-HTTP	219	[Psh,Ack] Seq=1500828308 Ack=2950738289	80	1580
12	18:10:47.152	202.181.96.37	192.168.0.20	TCP-HTTP	218	[Psh,Ack] Seq=1489984411 Ack=2950693243	80	1579
13	18:10:47.168	202.181.96.37	192.168.0.20	TCP-HTTP	218	[Psh,Ack] Seq=1489984589 Ack=2950693619	80	1579
14	18:10:47.168	202.181.96.37	192.168.0.20	TCP-HTTP	218	[Psh,Ack] Seq=1500828487 Ack=2950738672	80	1580

選択したパケットの同一プロトコル、同一送信元 IP, ポート、同一宛先 IP, ポートの組み合わせのデータを FIN まで検索し、パケットのうちデータ部のみを結合し、バイナリエディタ内で結合表示します。



画面上部に結合条件が表示されます。

左側が結合データの元となった分割されたパケットの一覧です。右側が結合されたデータです。

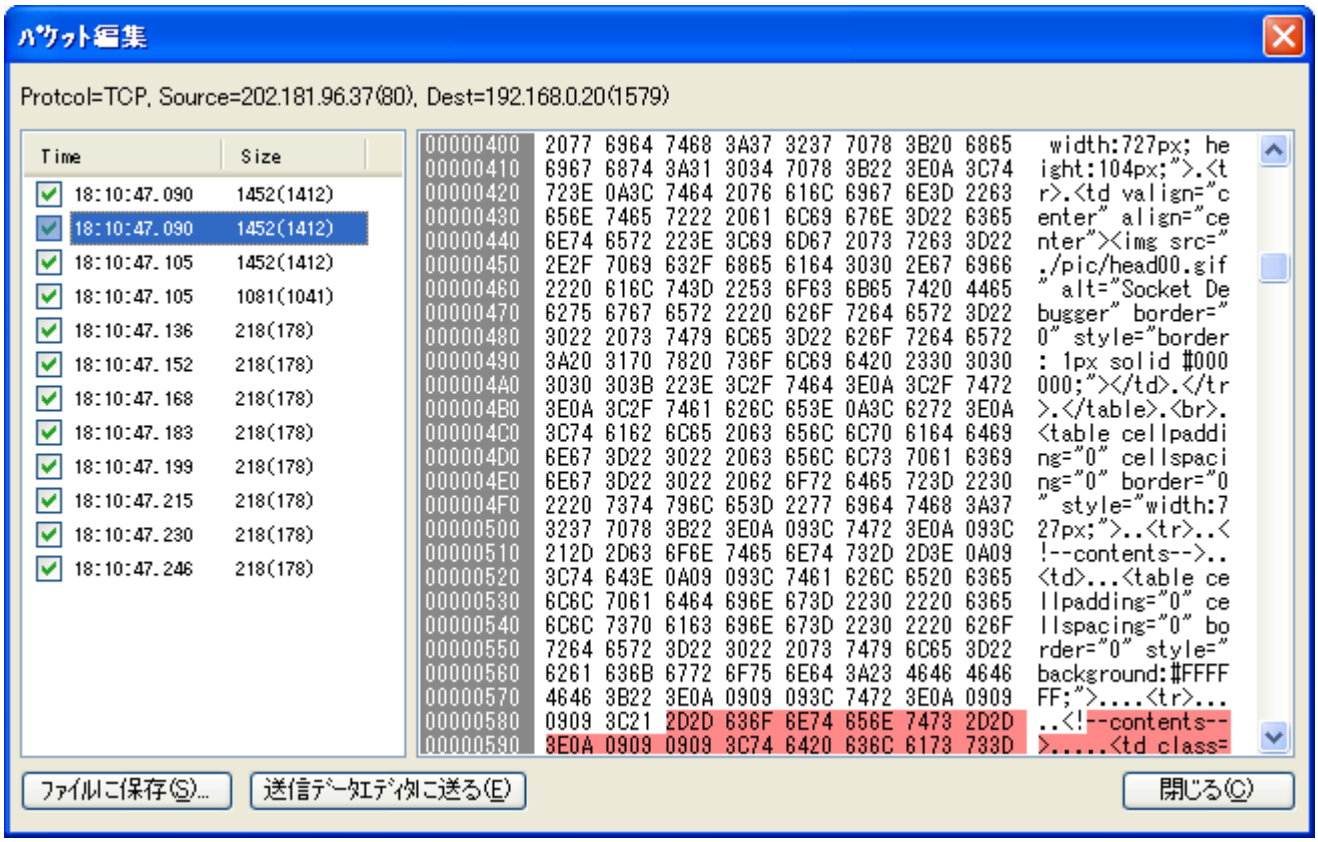
分割パケットはデータ時刻と、サイズ、(データ部サイズ) が表示されます。

分割パケットを選択すると、右側のデータ部で該当するデータエリアの背景を赤色表示し、

先頭データが表示されるように自動スクロールします。

チェックボックスを外すことで、結合データの中から個別に削除することが可能です。

（バイナリエディタ内の情報を直接編集することはできません）



バイナリエディタ内に表示されている情報は、名前を付けてファイルに保存や、

そのまま送信データエディタにセットすることが可能です。

※結合条件(TCP)

1．選択されたパケットのプロトコル、送信元 IP、ポート、宛先 IP、ポートを FIN まで取得します。
2．上記で抽出したパケットから、データ部のあるパケットをさらに抽出します。
3．抽出されたパケットのシーケンス番号を調査し、順番にソートします。

※結合条件(UDP)

1．選択されたパケットの同一シーケンス番号のパケットを抽出します。
(SocketDebgger のパケットモニタでは、UDP パケットは分割受信しないため意味のない機能のようですが、Ethereal などから読み込んだデータの場合に活用できます。)

ツール

ここではSocketDebuggerの基本設定と、各種ネットワークコマンドツールやデバイスツール、操作支援のダイアログなどについて使い方を説明します。

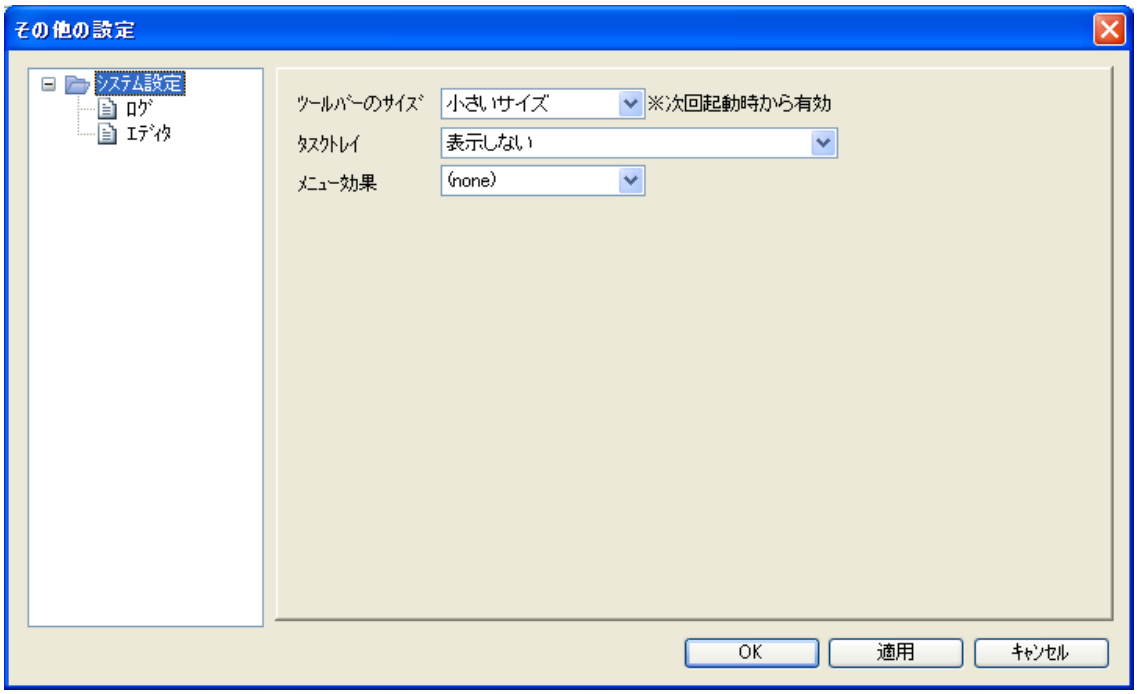
５－１．その他の設定

本機能は SocketDebugger の基本動作についての設定を行います。
呼び出すには SocketDebugger のメニューから「設定」－「その他の設定」を選択して下さい。

画面のデザイン・ウィンドウの挙動、メニューの表示方法、
扱うデータのサイズや自動保存、
バイナリエディタ部の表示などについてカスタムを行うことができます。
本画面で指定した内容は自動で保存され、次回起動からも有効です。

システム設定

左側の項目ペインからシステム設定を選択します。
ここでは、SocketDebugger のウィンドウ制御について設定を行うことが可能です。

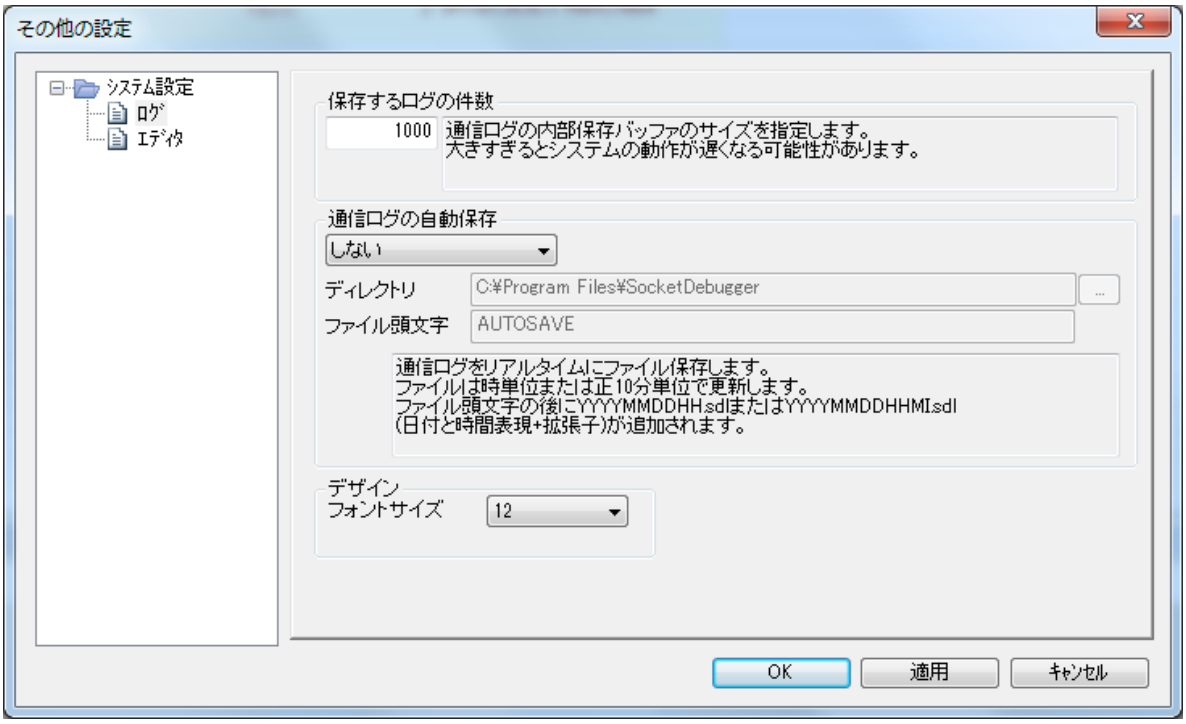


項目解説

ツールバーのサイズ	メインウィンドウのツールバーのサイズを２つ選択できます。 次回の起動から有効となります。 （小さいサイズ） （大きいサイズ）
タスクトレイ	タスクトレイ（及びタスクバー）での動作を指定します。 タスクトレイのアイコンはダブルクリックでウィンドウ表示、 右クリックでコンテキストメニューの表示を行えます。 「表示しない」 常にタスクバーに表示、タスクトレイアイコンは常に非表示です。 「表示する（タスクバー表示）」 常にタスクバー、タスクトレイアイコンは表示されます。 「表示しない（アイコン時タスクバー非表示）」 常にタスクトレイアイコンが表示され、 タスクバーからはウィンドウが最小化した場合に非表示となります。 タスクバー、タスクトレイの表示イメージ
メニュー効果	メニュー、コンテキストメニューの表示アニメーションを 選択することが可能です。

ログ

メインウィンドウで表示している通信ログのスペックや、自動保存、表示フォントサイズなどについて設定を行います。



項目解説

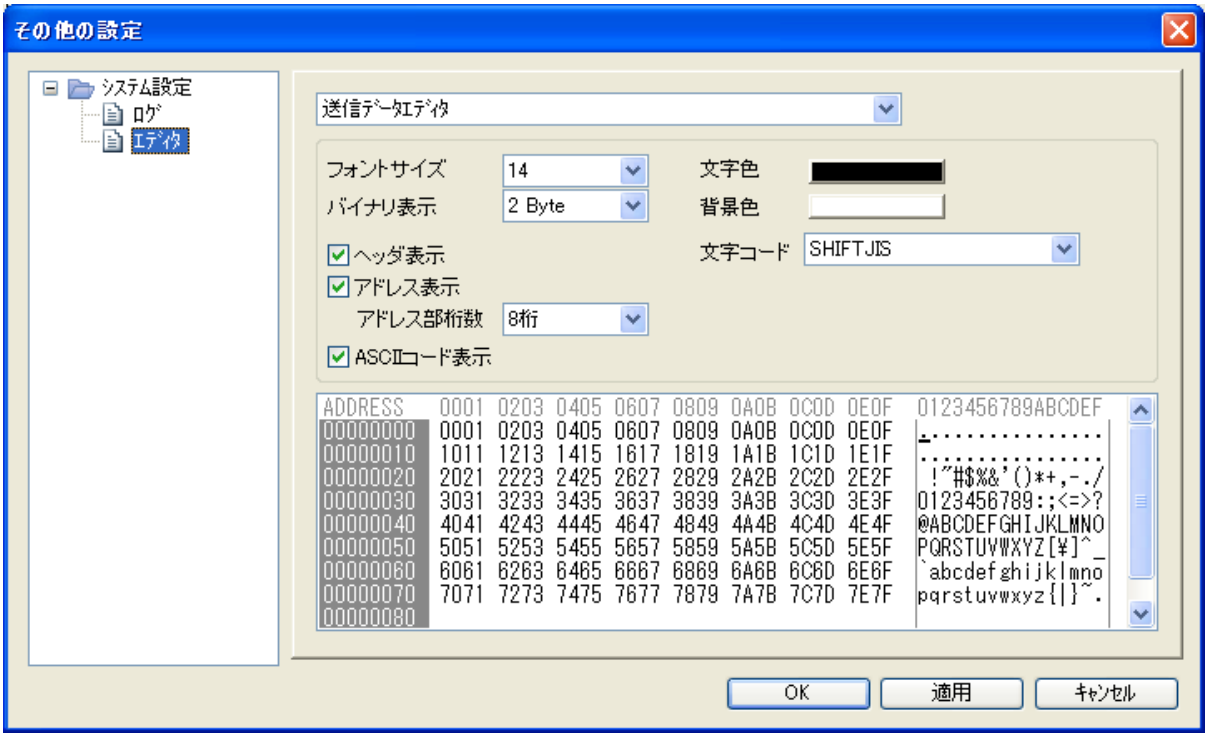
保存するログの件数	メインウィンドウに表示するログの件数を指定します。 入力範囲は 10～9999 です。 保存してる通信ログが多い場合にプログラムの動作が遅くなる場合がありますので、プログラムの利用目的にあわせて調整してください。
通信ログの自動保存	通信処理を行い、メインウィンドウにログが追加された場合にリアルタイムでログ保存を行います。 ファイルを保存するディレクトリと保存するファイルの頭文字が設定できます。 例えば、上記画像で、2005 年 09 月 06 日 19 時台のデータは C:¥Program Files¥SocketDebugger¥AUTOSAVE2005090619.sdl として、随時作成、追加が行われます。 作成した通信ログファイルは、手動作成したログファイルと同様に読み込むことが可能です。 上記説明は“時単位”を選択した場合です。“１０分単位”を選択した場合はファイル名が AUTOSAVE200509061910.sdl など、正分の分が挿入されます。
デザイン	メインウィンドウのログ表示用 View のフォントサイズを選択します。

送信データエディタ、詳細ログデータ、COM 制御１、COM 制御２

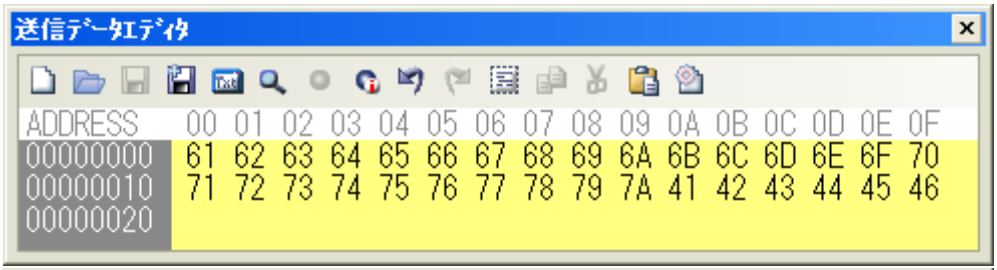
上記、４つのバイナリエディタの表示デザインについて設定を行います。

（SocketDebuggerFree では、COM 制御１，２はありません）

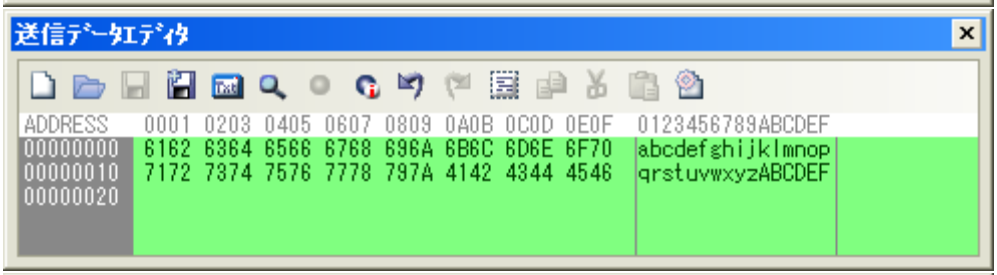
設定した内容は画面下部のサンプルプレビューに即時反映されます。



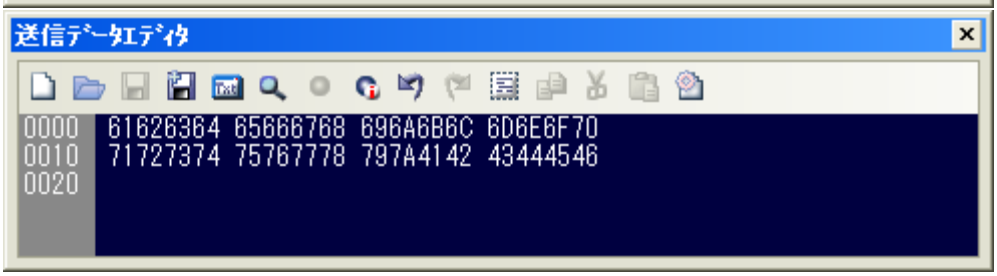
設定サンプル



フォント　：16
ヘッダ表示　：有
アドレス表示　：有 8 桁
バイナリ表示　：1 Byte
ASCII コード　：無
文字　：黒
背景　：薄黄色



フォント　：12
ヘッダ表示　：有
アドレス表示　：有 8 桁
バイナリ表示　：2 Byte
ASCII コード　：有
文字　：緑
背景　：薄緑



フォント　：14
ヘッダ表示　：無
アドレス表示　：有 4 桁
バイナリ表示　：4 Byte
ASCII コード　：無
文字　：白
背景　：濃青

項目解説

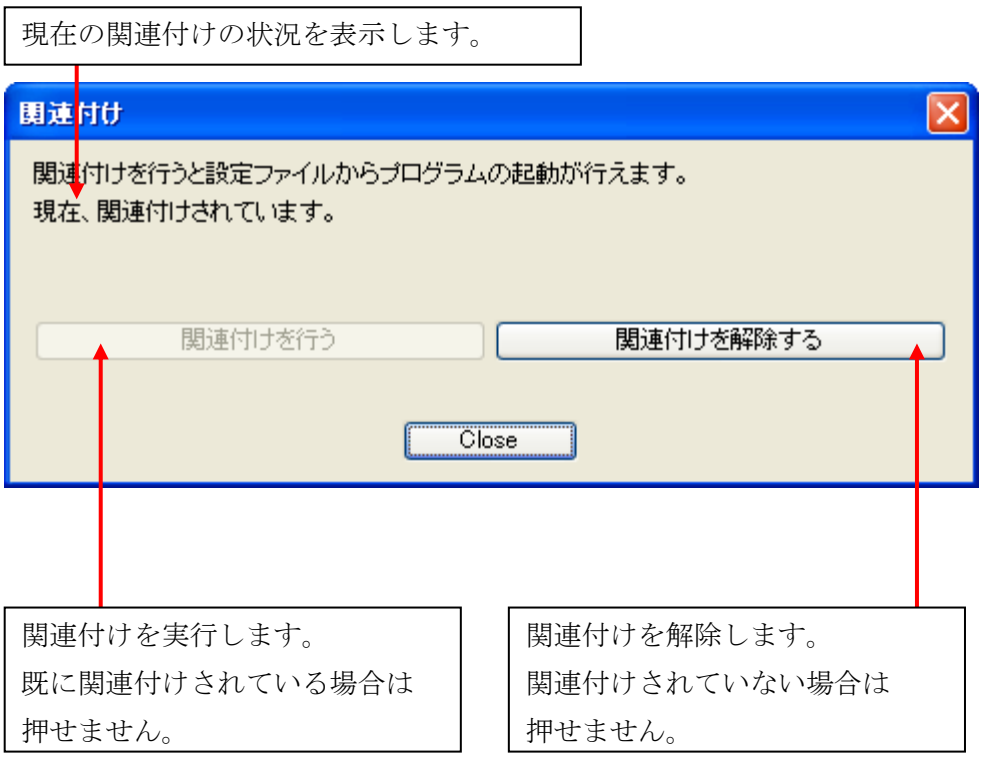
エディタ選択	設定するエディタを選択します。
フォントサイズ	テキストの表示サイズを設定します。 選択範囲は、１２，１４，１６，１８です。
ヘッダ表示	アドレスヘッダ部の表示の有無を設定します。
アドレス表示	データのアドレス部の表示の有無を設定します。
アドレス部桁数	アドレス部表示が有の場合のみ有効となります。 表示桁数を４桁、８桁で設定可能です。
バイナリ表示	バイナリ表示部のバイト区切りについて設定します。 １Byte，２Byte，４Byte で設定可能です。
ASCII コード表示	右側のアスキー表示の有無を設定します。
文字色	バイナリ部、アスキー部の文字の色を設定します。 色選択ダイアログが表示されます。
背景色	バイナリ部、アスキー部の背景の色を設定します。 色選択ダイアログが表示されます。
文字コード	テキスト部分の漢字コードの変換コードを指定します。 ASCII(変換なし)、SHIFTJIS、JIS、EUC、UTF8 から選択可能です。 この選択により、IME からの漢字入力で自動変換されます。 　JIS コードは 1B 24 42 コードが無いと漢字表示されません。 　JIS コードは半角カナが入力できません。
サンプルプレビュー	現在の設定内容の反映後の状態をプレビュー表示します。

５－２．関連付け

通信設定ファイル(.sdg)、通信ログファイル(.sdl)の２つのファイルについて
アプリケーションとの関連付けをレジストリに行い、設定ファイル（またはログファイル）から
アプリケーション起動を行えるように設定します。
逆に関連付けの解除も可能です。

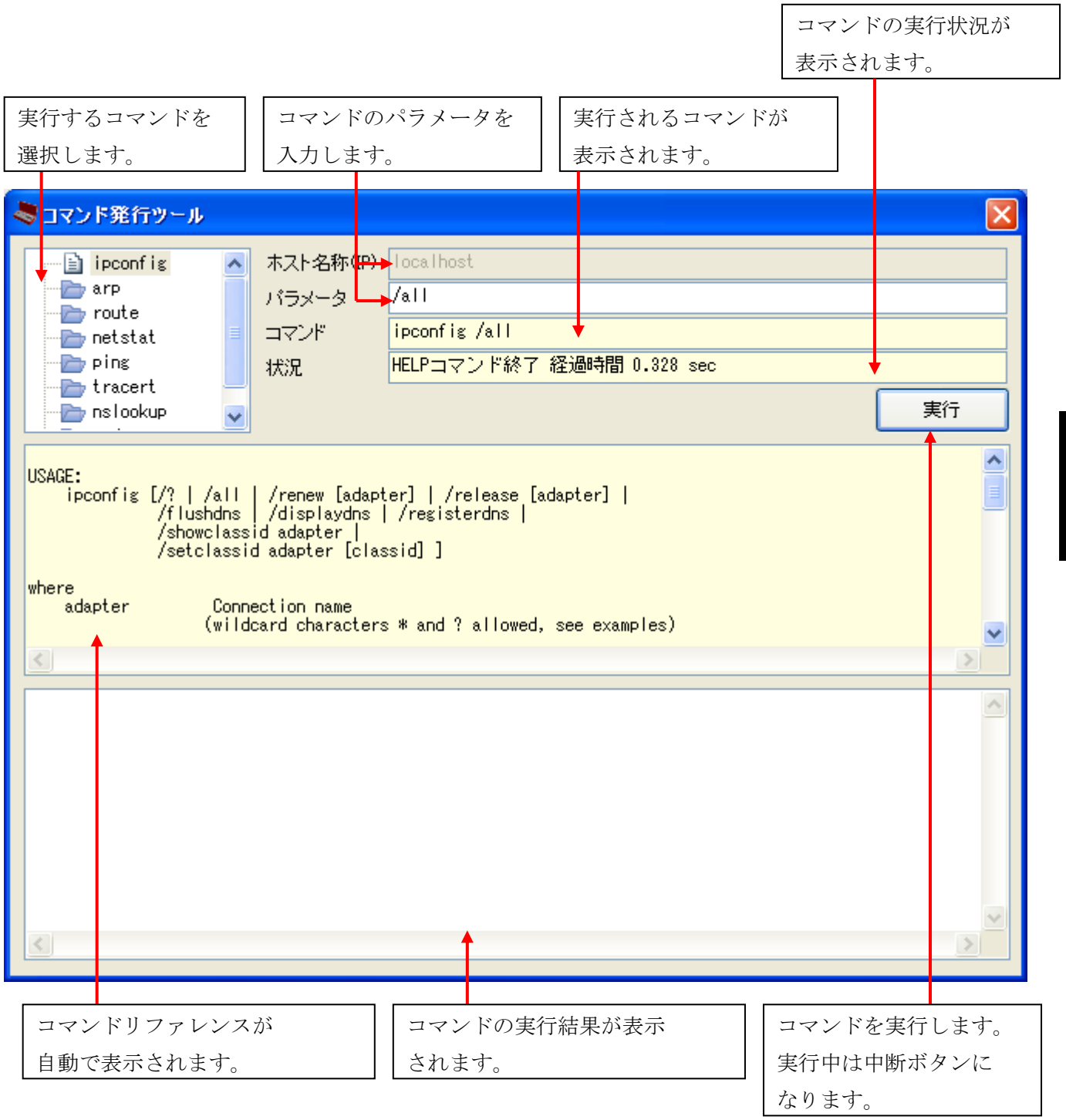
通信設定の保存ファイル(.sdg)

通信ログファイル(.sdl)

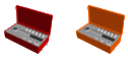
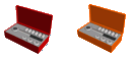
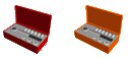
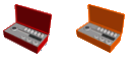
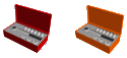
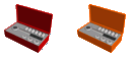
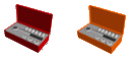


５－３．コマンド発行ツール

通常、コマンドプロンプトから行う各種のネットワーク用コマンド、デバイスコマンドを
本ダイアログから実行を行えます。入力支援がついているため簡単に操作を行うことができます。

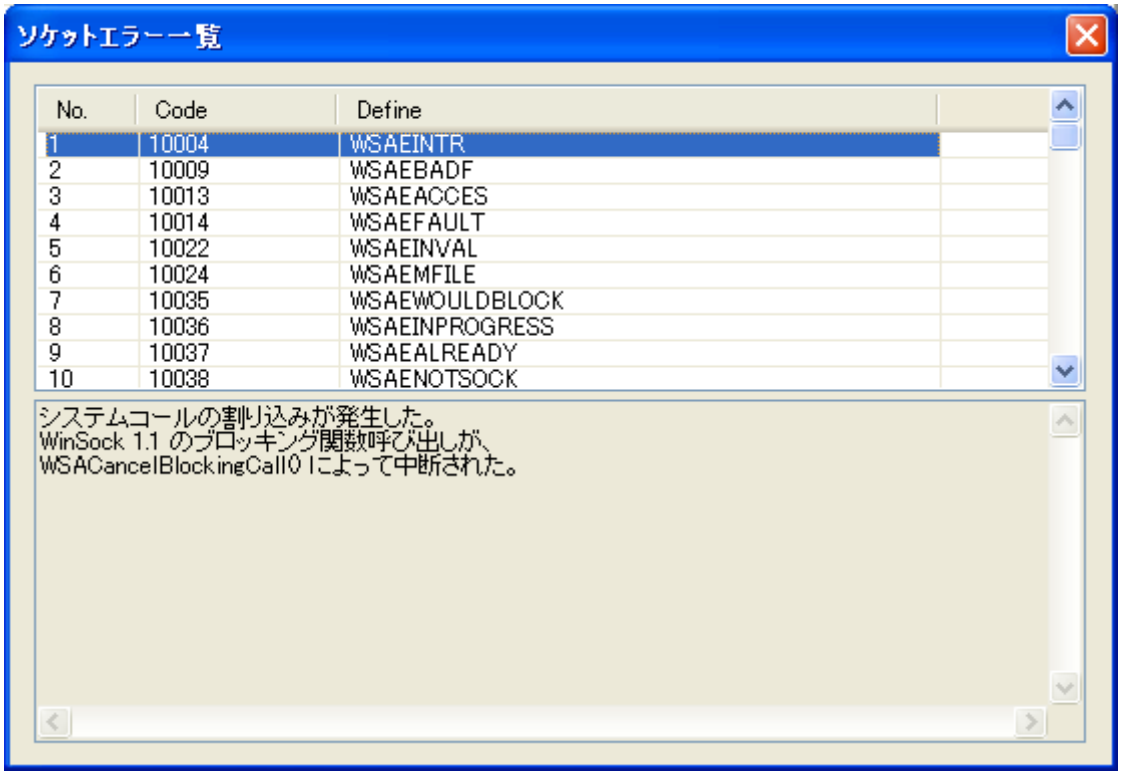


コマンド解説

ipconfig 	ネットワーク構成の表示および更新 自端末の IP アドレスの調査や再取得などに利用します。
arp 	ARP キャッシュの表示および変更 IP アドレスと MAC アドレスの認識状況の確認などに利用します。
route 	IP ルーティングテーブルの表示 ルーティング情報の表示や、端末に直接スタティックルートを設定する際に利用します。
netstat 	ネットワーク統計情報の表示 自端末が利用しているポートの状況確認などに利用します。
ping 	ip ネットワーク接続の確認 端末から、別端末までネットワークが接続されているか確認を行う際などに利用します。
tracert 	通信経路の確認 ルータなどで接続された大きなネットワークにおいて目的端末までのルータ経路の調査などに利用します。
nslookup 	DNS ネームサーバ情報の表示 ホスト名称から IP アドレスの取得を行います。 ホストの名前解決のテストなどに利用します。
mode	システムデバイスのモード表示 COM ポートのデバイス設定状況の表示、変更に利用します。

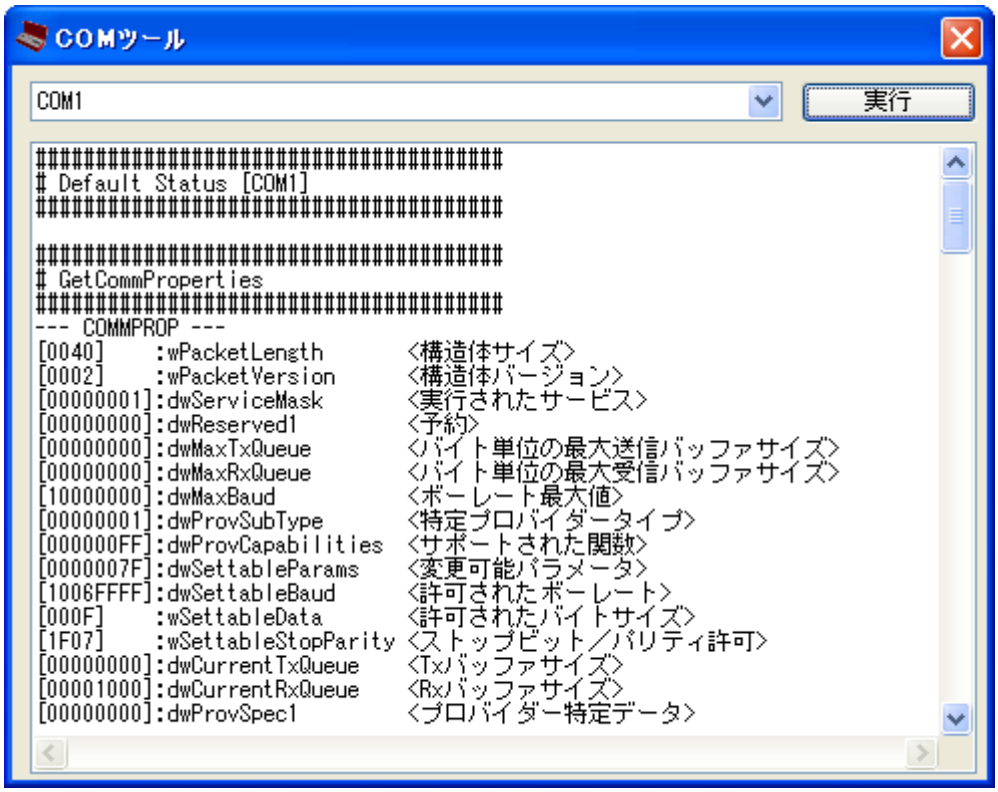
５－４．ソケットエラー一覧

Winsock API における、エラーコードと、その宣言名の一覧を表示し、
選択を行うことでそのエラーついての内容、原因などを表示します。
Socket プログラミングのデバッグにおいて利用下さい。



５－５．ＣＯＭツール

選択したCOMポートについて、シリアルデバイスに関する状況取得のAPI関数を実行し、
返された情報について表示を行います。
シリアルを利用したプログラミングのデバッグにご利用下さい。



実行される関数とその概略

GetCommProperties	デバイスの通信プロパティの取得 返される情報（構造体）：COMMPROP
GetCommState	デバイスの現在の制御設定の取得 返される情報（構造体）：DCB
GetCommMask	デバイスのイベントマスクの取得 返される情報（構造体）：DWORD
GetCommModemStatus	モデムの制御レジスタ値の取得 返される情報（構造体）：DWORD

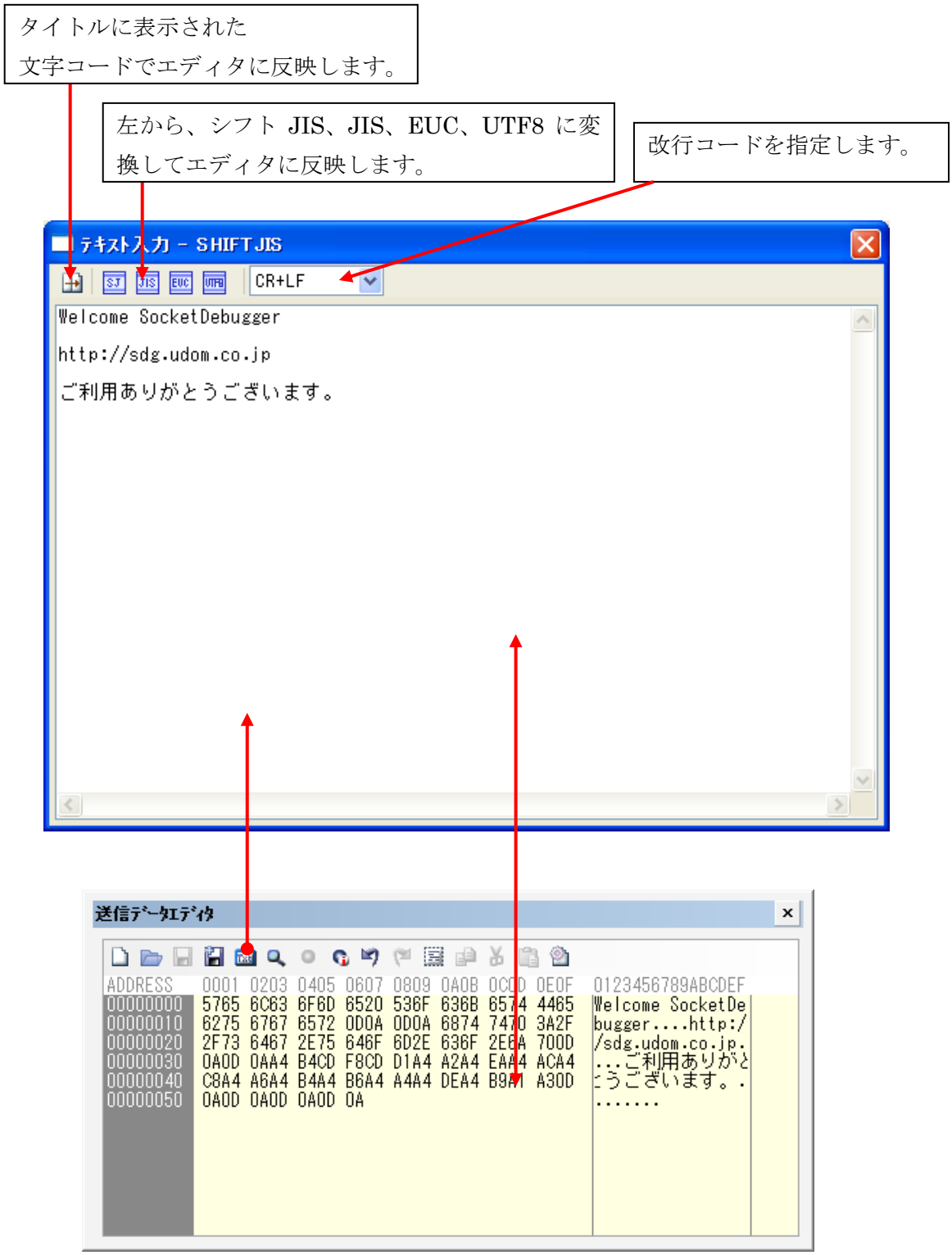
５－６．バージョン情報

各アプリケーション別に下記情報を表示します。

SocketDebugger	バージョン表示 サポートページ表示 レジスト情報 ソケット情報の表示 著作権表示
SocketDebuggerFree	バージョン表示 サポートページ表示 ソケット情報の表示 著作権表示
SerialDebugger	バージョン表示 サポートページ表示 著作権表示

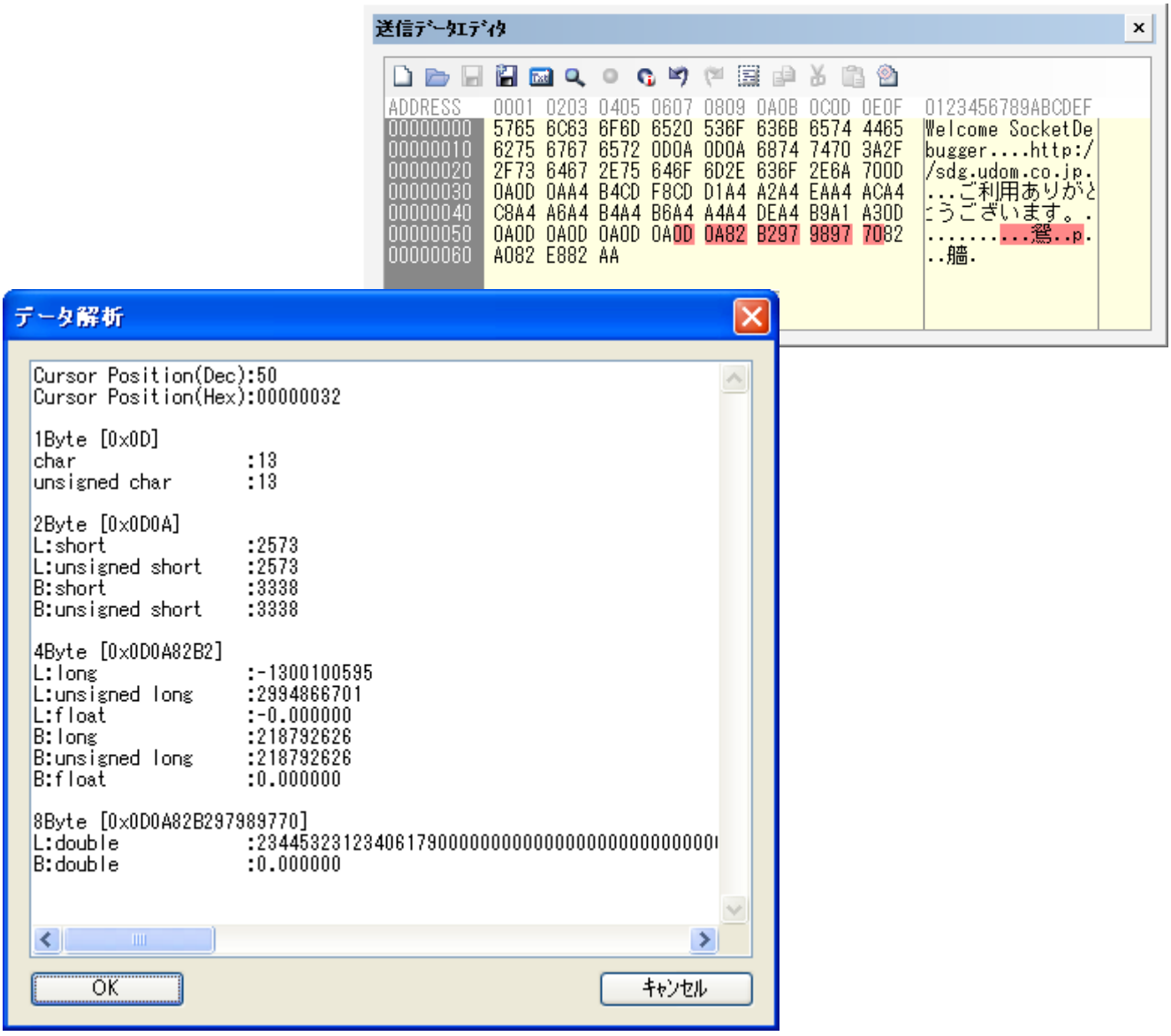
５－７．テキスト入力ダイアログ

送信データエディタの入力支援としてテキスト入力ダイアログを表示します。
表示時、既に送信データエディタ内にデータがる場合はそのデータが選択中の文字コードに変換されて表示されます。変換不可能なコードがあった場合、アドレスとともにエラーを表示します。
本画面から、任意の文字コードへの変換も可能です。



５－８．データ解析ダイアログ

バイナリエディタにおいて、カーソル位置からの８バイトを対象に
各種データ型として見た場合の一覧を表示します。
バイナリデータ内の対照のデータは赤く表示されます。



※注意
２Ｂｙｔｅ、４Ｂｙｔｅ、８Ｂｙｔｅの各項目の前のＬ：Ｂ：は
Ｌ：リトルエンディアン　Ｂ：ビッグエンディアンを表現しています。

その他

ここでは、ここまでで説明できなかったその他の機能についての補足説明をします。

６－１．機能比較表

項目	SocketDebugger	SocketDebuggerFree	SerialDebugger
アイコン			
ソフトウェア分類	シェアウェア	フリーソフト	フリーソフト
機能	●レジスト後有効　○機能あり　×機能なし		
複数起動	●	×	×
通信設定ファイルの作成	●	×	×
通信設定ファイルの読み込み	●	auto.sdg のみ	auto.sdg のみ
ファイルの関連付け	●	×	×
ソケット機能	○	○	×
シリアル機能	○	×	○
パケットキャプチャ機能	●	×	×
コマンド発行ツール	○	ソケットのみ	シリアルのみ
ソケットエラー一覧	○	○	×
UI の変更	○	×	×
ウィンドウ			
メインウィンドウ	○	○	○
送信データエディタ	○	○	○
詳細ログデータ	○	○	○
接続状況	○	○	○
送信データエクスプローラ	○	○	○
ＣＯＭポート制御１，２	○	×	○
スクリプトエディタ	○	○	○
パケットキャプチャウィンドウ	○	×	×
パケットヘッダ	○	×	×
バイナリデータ	○	×	×

６－２．バイナリエディタ機能詳細

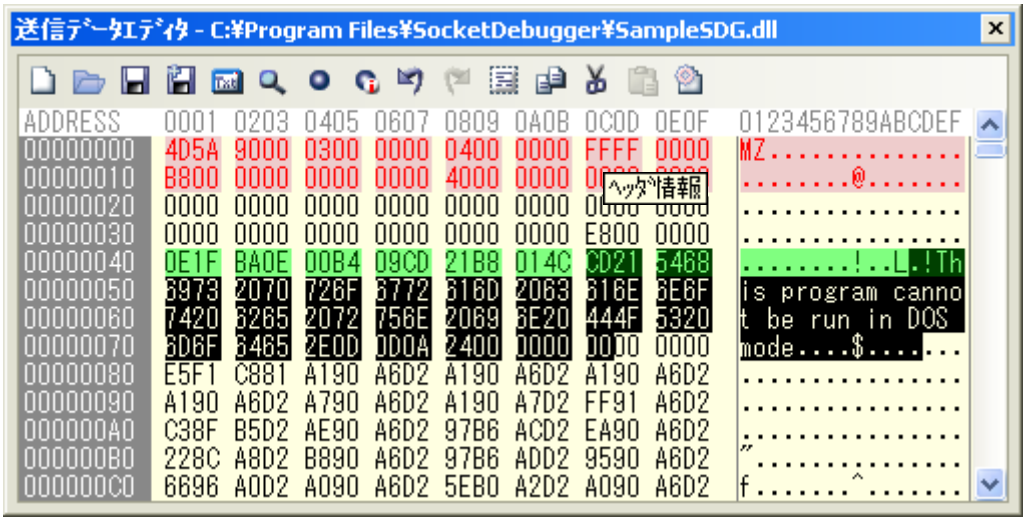
バイナリ部 受付キー：0～9、a～f、A～F
アスキー部 受付キー：制御コード以外のアスキーコード 全角入力：IME から入力可能 　漢字コードは現在、選択されている文字コードに変換されて入力されます。 　JIS コードは漢字コード開始の 1B 24 42 コードが無いと漢字表示されません。 　JIS コードは半角カナの入力が行えません。

その他キー操作		
キー	挿入モード	上書きモード
Ctrl + A	全て選択	←
Ctrl + Z	ひとつ前に戻す（Undo）	←
Ctrl + X（選択中） Shift + Delete（選択中）	切り取り	コピー＋範囲内０クリア
Ctrl + C（選択中） Ctrl + Insert（選択中）	コピー	←
Ctrl + V Shift + Insert	貼り付け(挿入)	貼り付け（データサイズ内で可能なサイズ分上書き）
Ctrl + V（選択中）	貼り付け(選択範囲削除して)	貼り付け（選択範囲内のみ）
Insert	上書きモードへ	挿入モードへ
BackSpace	カーソル前の文字削除	無効
BackSpace(選択中)	選択データ削除	無効
Delete	カーソルの文字削除	無効
Delete(選択中)	選択データ削除	範囲内０クリア

キー	挿入モード	上書きモード
カーソルキー（↑）	カーソル移動	←
カーソルキー（↓）	カーソル移動	←
カーソルキー（←）	カーソル移動	←
カーソルキー（→）	カーソル移動	←
Shift + カーソルキー（↑）	カーソル移動＋選択	←
Shift + カーソルキー（↓）	カーソル移動＋選択	←
Shift + カーソルキー（←）	カーソル移動＋選択	←
Shift + カーソルキー（→）	カーソル移動＋選択	←
ホイール	３行上下スクロール	←
Home	カーソル行の先頭へ移動	←
End	カーソル行の最後へ移動	←
Ctrl + Home	先頭へ移動	←
Ctrl + End	最後へ移動	←
Shift + Home	カーソル行から同一行の先頭までを選択	←
Shift + End	カーソル行から同一行の最後までを選択	←
PageUp	１ページ分、上にカーソル移動	←
PageDown	１ページ分、下にカーソル移動	←
Shift + PageUp	カーソル行から 1 ページ分上までを選択	←
Shift + PageDown	カーソル行から 1 ページ分下までを選択	←

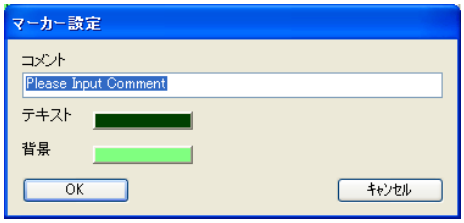
マーカーについて

送信データエディタ、詳細ログデータ画面では、バイナリエディタにマーカー情報の設定が可能です。
マーカー情報を設定すると、データのエリア情報の見分けが付くようになり非常に便利です。



マーカーの設定

設定したい領域をドラッグなどで選択し、ツールバー、またはコンテキストメニューの
マーカーを選択してください。テキスト色、背景色、ツールチップコメントの入力ダイアログが
表示されます。



マーカーの削除

既にマーカーとして設定されている領域を選択、マーカーを選択してください。
削除の確認ダイアログが表示されます。

マーカー情報

ツールバー、またはコンテキストメニューからマーカー情報を選択します。

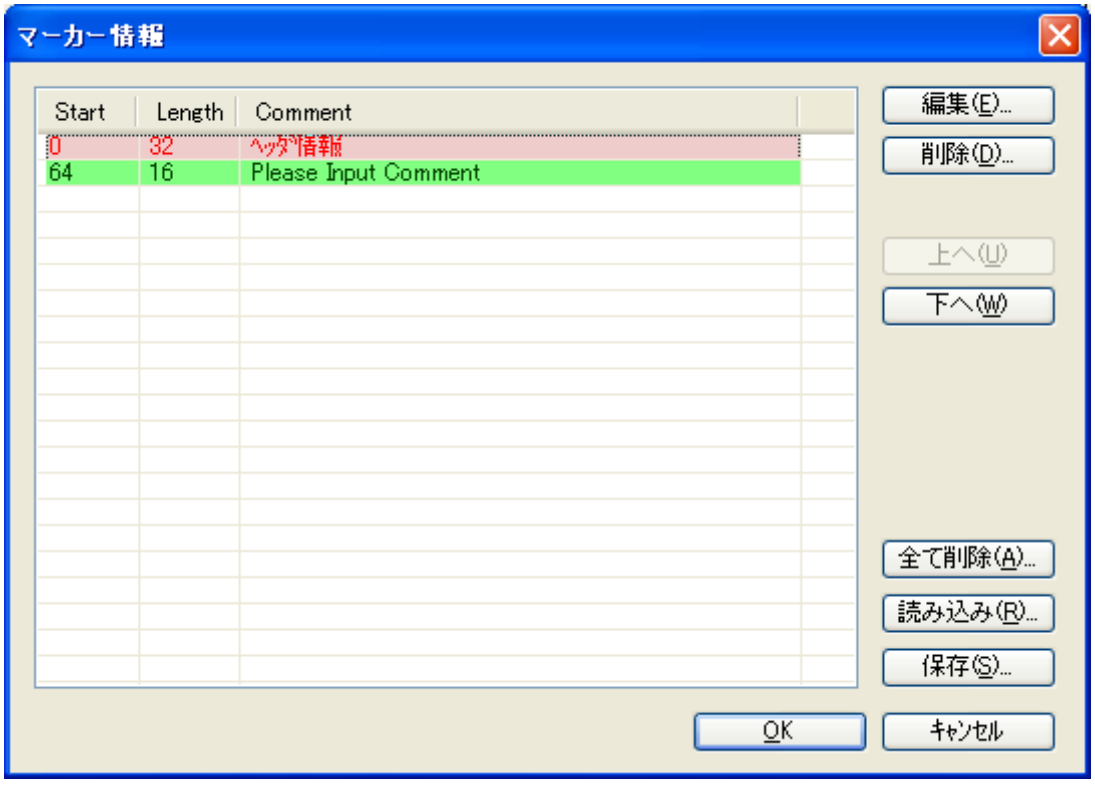
下記、マーカー情報ダイアログが表示されます。

ここでは、既に登録してあるマーカー設定の

テキスト色、背景色、ツールチップコメントの変更、

削除、順番の入れ替え（順番を替えても表示に変わりはありません。管理上だけです。）

マーカー設定のファイル保存、読み込みが行えます。



※マーカー情報は、通信設定と共に通信設定ファイル（.sdg）にも保存されます。

６－３．拡張ＤＬＬの作り方

拡張組み込みＤＬＬについて

ユーザ固有の通信処理仕様によるテストを行うにあたって、通常の設定だけでは実現不可能な
独自のビジネスロジックを DLL という形で組み込むことが可能です。

例えば、ある通信伝文を受信した際に、その内容に応じた応答伝文を返したり、他のサブシステムのもつ
データを参照して、その結果を応答したりすることができます。

拡張 DLL は SocketDebuggerFree,SerialDebugger も利用できます。

組み込みＤＬＬの概要

DLL 内部で、1 本のコールバック用関数を用意します。SocketDebugger 本体から状況によりコールされ、
データの受け渡しや、状況に応じた挙動を指定できます。
また、DLL 内で設定ウィンドウを表示し、その設定内容を本体に渡すことにより本体の通信設定と同様に
設定の保存が可能となり、プロジェクトに応じた設定可能な拡張処理の作成を行うことができます。

組み込みＤＬＬの作成方法

SocketDebugger に組み込み可能な DLL の作成方法について記述します。

※環境については Visual C++ 6.0 で紹介します。

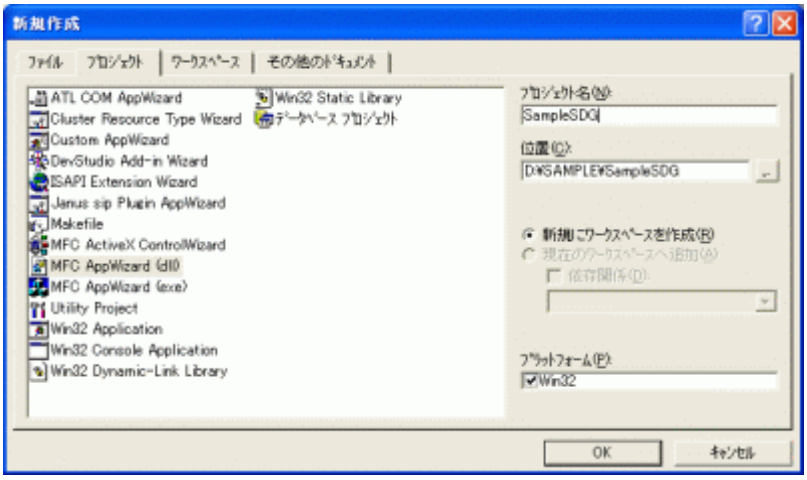
①プロジェクトを作成する。

Visual C++を起動し、「ファイル」→「新規作成」で

プロジェクトタブを選択します。

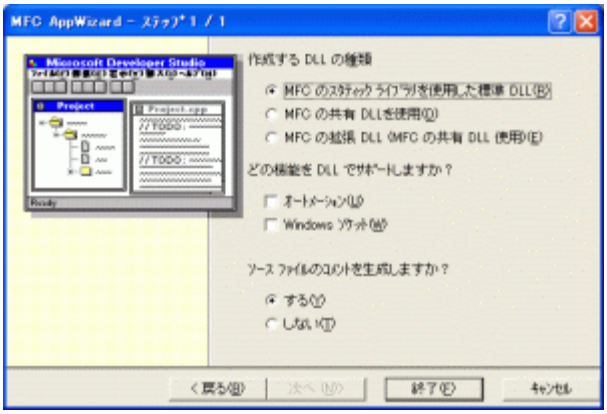
MFC AppWizard (dll)を選択し、dll の名前を入力します。

（ここでは SampleSDG としますが、自由に宣言ください。）



②設定ウィザード　ステップ 1/1

DLL の設定で　MFC のスタティックライブラリを指定します。



③定義ファイルのインクルードとコールバック関数の記述

SDGDLLEX.H をサポートページよりダウンロードします。

※SDGDLLEX.H 内部にも使い方の説明がありますのでそちらも参照下さい。

SampleSDG.cpp を開いて、SDGDLLEX.H をインクルードします。

```
// SampleSDG.cpp : DLL 用の初期化処理の定義を行います。
//

#include "stdafx.h"
#include "SDGDLLEX.H"
#include "SampleSDG.h "
```

コールバック関数の雛形を作成します。

SampleSDG.cpp の最後に追加します。

```
////////////////////////////////////////////////
// 唯一の CSampleSDGApp オブジェクト

CSampleSDGApp theApp;


//-----------------------------------------------
//      SDG コールバック関数
//-----------------------------------------------
DWORD WINAPI CallBackSDG( ULONG dest, ULONG src ){
    return SDGDLL_RESULTOK;
}
```

SampleSDG.def ファイルにコールバック関数を定義します。

```
; SampleSDG.def : DLL 用のモジュール パラメータ宣言
LIBRARY      "SampleSDG"
DESCRIPTION  'SampleSDG Windows Dynamic Link Library'
EXPORTS
    ; 明示的なエクスポートはここへ記述できます
    CallBackSDG
```

以上でプログラム環境的な設定は終了です。

まずはここまででコンパイルが通るか確認してください。

④次にここでの内部処理の仮仕様を決めます。

１．設備サーバと複数のＰＣ間で８バイト固定サイズで
　　データ伝送を行う。
２．データ伝送が起きるパターンはＰＣから、
　　クライアントでコネクションを行い
　　設備サーバに設備の動作指示の通信を行う。
　　＠ 機械Ａを稼動--> "A0000000" (Ascii)
　　＠ 機械Ａを停止--> "A0000001" (Ascii)
　　この要求に対し、設備サーバでは
　　要求に対する処理を行ったのち、
　　＠ 正常終了--> "OK" (Ascii)
　　＠ 異常終了--> "ERR" (Ascii)
　　を返送する。ただし、ＰＣからの通信内容が誤りの場合も
　　異常終了コードを返送する。
３．この設備サーバの動作をシミュレートする。

上記仕様をプログラムにします。
（プログラムの内容は次紙に示します。）

次紙プログラムをコピーしてビルドしてください。
SocketDebugger と同じフォルダに格納して、設定ダイアログから
拡張ＤＬＬを選択すれば、実行可能です。
動作設定で拡張動作を選択する必要があります。

ここで作成したサンプルは、ダウンロードページに格納します。
最も簡単な応答プログラムとして活用いただけるのではないかと思います。
ダウンロードページには他にも有用なサンプルがソース付きで公開されていますので
サンプルとヘッダを参照して、対象システムにあった拡張ＤＬＬを作成してください。

プログラムサンプルソース

```
//////////////////////////////////////////////////////////////////////////////
// 唯一の CSampleSDGApp オブジェクト

CSampleSDGApp theApp;

#define SDGDLL_VER      (0x100)              // DLL バージョンの宣言
MC_SDGMODULE;                           // 一般的なサブルーチンの宣言
//---------------------------------------------------------------
//      DLL 説明文の宣言
//---------------------------------------------------------------
MC_SDGHEAD(
_T("通信応答テストプログラム"),
_T("データを受信したら要求コードをチェックして¥r¥n")
_T("OK、ERR を返送します。¥r¥n")
);
//-----------------------------------------------
//  SDG コールバック関数
//-----------------------------------------------
DWORD WINAPI CallBackSDG( ULONG dest, ULONG src ){
    MC_MODULEUSE;                           // 変数宣言
    //-------------------------------------------
    //  定型マクロ処理
    //-------------------------------------------
    MC_EV_DEFAULT;                          // 基本パラメータセット
    MC_EV_GETCOMMENT;                       // DLL のコメント情報を取得する
    MC_EV_GETVER;                           // ver.を取得する
    MC_EV_CONFIGRESULT(SDGDLL_CONFIGNOT);   // コンフィグの有無を取得する
    //-------------------------------------------
    //  データ受信
    //-------------------------------------------
    if( srcData->command & SDGDLL_ONRECEIVE ){
        BOOL    bResult = FALSE;            // はじめはエラー返送をセット
        if( srcData->buffsize == 8 ){       // 受信データは規定のサイズ？
            if( memcmp( srcData->lpbuffer, _T("A0000000"), 8 ) == 0 ||
                memcmp( srcData->lpbuffer, _T("A0000001"), 8 ) == 0 ){
                bResult = TRUE;             // OK の返送を行う
            }
        }
        destData->command = SDGDLL_REQSEND;     // データ送信要求を行う
        destData->wparam = 0;                   // 受信したポートのみ返送する
        destData->lpbuffer = srcData->lpbuffer; // 受信データエリアを
                                                // 送信データエリアに流用
        // ここで、受信データエリアは通信の設定で決めた
        //「内部プログラム受信バッファサイズ」が適用されていますので、
        // そのサイズまでは上書きが可能になります。
        // サイズがそれを超過する可能性がある通信仕様では、
        // 内部で確保する必要があります。
        // 確保した場合は開放のタイミングを作成することをお忘れなく
        if( bResult ){
            destData->buffsize = 2;             // 送信データは 2Byte
            lstrcpy( (char*)destData->lpbuffer, _T("OK") );
        }
        else{
            destData->buffsize = 3;             // 送信データは 3Byte
            lstrcpy( (char*)destData->lpbuffer, _T("ERR") );
        }
        return SDGDLL_RESULTOK;
    }
    return SDGDLL_RESULTOK;
}
```

６－４．Lua スクリプト拡張説明

Lua 拡張について

SocketDebugger では、通信処理をユーザが自由な仕様に従ってテストが行えるよう、
デフォルトの設定だけでは実現不可能なロジックを Lua スクリプトにより実現することが可能です。

例えば、ある通信伝文を受信した際に、その内容に応じた応答伝文を返したり、
一定時間で複数の違うデータを送信したりすることができます。
SocketDebuggerFree,SerialDebugger も利用可能です

Lua 拡張の概要

SocketDebugger は Lua スクリプトで記述されたいくつかのイベントを状況により呼び出します。
ユーザはその渡されたデータを判定し、用意された関数で通信などのアクションを起こすことが可能です。
スクリプト言語 Lua のリファレンスは下記などで検索してください。
Google　{Lua 5.1 リファレンス}　検索

Lua スクリプトはポート１、ポート２それぞれに記述できます。
それぞれの専用エディタから任意の処理を記述ください。
プログラムは通信データとともに保存されますので
別途ファイル保存等は必要ありません。
スクリプト動作を実行するには「設定」－「通信設定」の
任意ポートの「動作」から「スクリプト制御を行う」にチェックを付けてください。

Lua 拡張のイベントと関数

SocketDebugger で拡張した、イベントと関数について説明します。

■イベント一覧

通信などの状況により SocketDebugger 本体から下記の Lua 関数が呼び出されます。
ユーザは以下の関数内部を実装することにより任意のアクションが可能となります。

OnConneted	通信接続(または開始)時に発生
OnSendPush	送信ボタン押下時に発生
OnTimer	タイマーイベント時に発生
OnReceive	データ受信時に発生
OnDisConnected	通信切断(または終了)時に発生

※UDP,COM の通信では接続/切断は開始/終了となります。

■関数一覧

上記イベント関数内から呼び出し可能な SokcetDebugger に指示を行う拡張関数です。

Logput	ログ出力
Disconnect	通信切断
SendData	データ送信
SetTimer	タイマー起動
KillTimer	タイマー停止
GetEditorData	送信エディタデータ取得
SetEditorData	送信エディタデータ格納
FileRead	ファイル読み込み
FileWrite	ファイル書き込み
BitOr	ビット演算 OR
BitAnd	ビット演算 AND
BitOff	ビット操作 OFF

イベント（１／２）

接続完了通知

OnConnected()

<< 引数 >>
無し

<< 戻り値 >>
0 固定

<< 概要 >>
通信が確立した場合に発生します。
ただし、UDP/OCM 通信では、処理開始時に発生します。

送信ボタン押下

OnSendPush()

<< 引数 >>
無し

<< 戻り値 >>
0 固定

<< 概要 >>
プログラム本体の「送信ボタン」を押下時に発生します。
シーケンスにユーザ任意のタイミングを作成したい場合に利用して下さい。

タイマー通知

OnTimer(*id*)

<< 引数 >>
id　　[整数]
発生したタイマー番号
0 or 1

<< 戻り値 >>
0 固定

<< 概要 >>
ユーザが起動したタイマーイベントにより発生します。

イベント（２／２）

受信通知

OnReceive(*recv*)

<< 引数 >>

recv	[テーブル] 受信データ

<< 戻り値 >>
0 固定

<< 概要 >>
データ受信時に発生します。
COM は、通信設定した受信データ区切りにより発生を制御します。

切断通知

OnDisConnected()

<< 引数 >>
無し

<< 戻り値 >>
0 固定

<< 概要 >>
TCP では通信の切断時、
UDP,COM では通信処理の終了時に発生します。

関数（１／５）

ログ出力

Logput(*infonumber*, *outstring*)

<< 引数 >>

infonumber	[整数] 出力するログのカテゴリ番号 1:情報 2:注意 3:警告
outstring	[文字列] ログ出力する文字列

<< 戻り値 >>
無し

<< 概要 >>
Lua スクリプト内から任意に通信ログに情報出力を行います。

通信切断

Disconnect()

<< 引数 >>
無し

<< 戻り値 >>
無し

<< 概要 >>
通信を切断します。
自接続ポートのみで、TCP クライアント,UDP,COM は通信処理自体の
終了となりますが、TCP サーバの場合、接続待機は終了しません。

データ送信

SendData(*tabledata*, *option*)
SendData(*stringdata*, *option*)

<< 引数 >>

tabledata	[テーブル] 送信するデータを指定します。 データはテーブルに格納された整数データ(0～255)で ある必要があります。
stringdata	[文字列] 送信するデータを指定します。 指定された文字列をそのまま送信します。
option	[整数:(省略可)] データ送信方法を指定します。 0:現在の制御中の接続ポートで送信します。 1:ポート１で接続中の全通信から送信します。 2:ポート２で接続中の全通信から送信します。 省略時は 0 指定となります。

<< 戻り値 >>
無し

<< 概要 >>
指定したデータを送信します。

関数（２／５）

タイマー起動

SetTimer(*timernumber*, *cycle*, *_1time*)

<< 引数 >>

timernumber	[整数] タイマー番号（タイマーは最大２コ迄） 0:タイマー１指定 1:タイマー２指定
cycle	[整数] イベントを発生する周期をミリ秒で指定する
_1time	[整数:(省略可能)] 0:通常タイマー 1:ワンタイムタイマー(１度イベント発生後、自動でタイマーを停止する) 省略時は 0:通常タイマーとなる。

<< 戻り値 >>
無し

<< 概要 >>
タイマーイベントを発生させる周期を指定し、タイマーを起動します。
タイマーは２つまでセットすることが可能です。
ワンタイムタイマーは、１度イベント発生後に自動で停止します。
通常タイマーはユーザが停止させるコードを記述する必要があります。
スクリプト制御の終了時は、稼動中のタイマーは自動で停止します。

タイマー停止

KillTimer(*timernumber*)

<< 引数 >>

timernumber	[整数] タイマー番号（タイマーは最大２コ迄） 0:タイマー１指定 1:タイマー２指定

<< 戻り値 >>
無し

<< 概要 >>
起動していたタイマーを停止します。

関数（３／５）

送信エディタデータ取得

editdata = GetEditorData()

<< 引数 >>
無し

<< 戻り値 >>
editdata　　[テーブル]
　　取り出した送信エディタデータ

<< 概要 >>
送信エディタに格納されているデータを取得します。

送信エディタデータ格納

SetEditorData(*editdata*)

<< 引数 >>
editdata　　[テーブル]
　　格納する送信エディタデータ

<< 戻り値 >>
無し

<< 概要 >>
送信エディタにデータを格納します。

ファイル読み込み

data = FileRead(*filename*)

<< 引数 >>
filename　　[文字列]
　　読み込むファイル名称

<< 戻り値 >>
data　　[テーブル]
　　読み出したデータ
　　読み込みエラー時は null が返る

<< 概要 >>
指定したファイルを読み込み、テーブル型で取得します。

ファイル書き込み

FileWrite(*data*,*filename*)

<< 引数 >>
data　　[テーブル]
　　書き込むデータ
filename　　[文字列]
　　書き込むファイル名称

<< 戻り値 >>
無し

<< 概要 >>
指定したファイルにデータを書き込みます。

関数（４／５）

ビット演算 OR

ans = BitOr(*base*, *param*)

<< 引数 >>

base	[整数(1 バイト)]
	OR を行う右値を指定します。
param	[整数(1 バイト)]
	OR を行う左値を指定します。

<< 戻り値 >>

ans	[整数(1 バイト)]
	base,param の OR 値を返します。

<< 概要 >>

指定した２つの１バイトの値の OR 値を返します。

ビット演算 AND

ans = BitAnd(*base*, *param*)

<< 引数 >>

base	[整数(1 バイト)]
	AND を行う右値を指定します。
param	[整数(1 バイト)]
	AND を行う左値を指定します。

<< 戻り値 >>

ans	[整数(1 バイト)]
	base,param の AND 値を返します。

<< 概要 >>

指定した２つの１バイトの値の AND 値を返します。

ビット演算 BitOff

ans = BitOff(*base*, *param*)

<< 引数 >>

base	[整数(1 バイト)]
	ビット操作の元の値を指定します。
param	[整数(1 バイト)]
	OFF を行うビットに１をたてた値を指定します。

<< 戻り値 >>

ans	[整数(1 バイト)]
	base から param のビットを OFF した値を返します。

<< 概要 >>

元の値から指定したビットを OFF した値を返します。

関数（５／５）

文字列からテーブル変換

ans = StringToTable(*string*)

<<引数>>
string（文字列）　　変換する文字列

<<戻り値>>
テーブル　　正常終了
　　変換ﾃﾞｰﾀが格納される
数値　　異常終了
　　1:パラメータエラー
　　2:文字列認識エラー

<<概要>>
渡された文字列をテーブル配列に格納する。

<<例>>
ans = FileReadEx("20071104")
└ans（テーブル)に ASCII 分解された文字列が格納される
　エラー発生時は result は数値型であり、エラーコードが格納される。

テーブル同士の結合

ans = TableCat(*table1*, *table2*)

<<引数>>
table1(テーブル)　　前結合データ
table2(テーブル)　　後結合データ

<<戻り値>>
テーブル　　正常終了
　　結合データが格納される
数値　　異常終了
　　1:パラメータエラー
　　2:前結合データ認識エラー
　　3:後結合データ認識エラー

<<概要>>
テーブル同士を結合してひとつのテーブルに編集する。

<<例>>
ans = TableCat(a, b)
└ans（テーブル)に結合されたデータが返る
　エラー発生時は ans は数値型であり、エラーコードが格納される。

スクリプトサンプル

例１．受信したデータを判定して内容にあったファイルデータを送信、または通信切断を行う。

```
-----------------------------------------------
-- 接続完了通知
-----------------------------------------------
function OnConnected()
    Logput(1,'OnConnected')
    return 0
end
-----------------------------------------------
-- 送信ﾎﾞﾀﾝ押下
-----------------------------------------------
function OnSendPush()
    Logput(1,'OnSendPush')
    a = GetEditorData()
    SendData(a)
    return 0
end
-----------------------------------------------
-- ﾀｲﾏｰ通知
-----------------------------------------------
function OnTimer(id)
    Logput(1,'OnTimer')
    return 0
end
-----------------------------------------------
-- 受信通知
-----------------------------------------------
function OnReceive(recv)
    Logput(1,'OnReceive')
    if recv[1] == 0x00 then
        -- 先頭が 0x00 ならば data1 を返す
        Logput(1,'DATA 1')
        s = FileRead( 'c:¥¥debug¥¥data1.bin' )
        SendData(s)
    elseif recv[1] == 0x01 then
        -- 先頭が 0x01 ならば data2 を返す
        Logput(1,'DATA 2')
        s = FileRead( 'c:¥¥debug¥¥data2.bin' )
        SendData(s)
    elseif recv[1] == 0xFF then
        -- 先頭が 0xFF ならば 通信切断
        Logput(1,'Disconnect')
        Disconnect()
    end
    return 0
end
-----------------------------------------------
-- 切断通知
-----------------------------------------------
function OnDisConnected()
    Logput(1,'OnDisConnected')
    return 0
end
```

この他にも、SocketDebugger WEB ページにはサンプルがありますので、そちらも参照ください。

６－５．Lua スクリプト命令拡張説明

Lua 命令拡張について

SocketDebugger では、ユーザが Lua スクリプトにより通信処理を
自由にカスタマイズすることが可能です。

ですが、スクリプトでは記述できない処理、または
ユーザのサブシステムと連動する必要がある処理などは
標準のままでは表現できません。

SocketDebugger では、Lua スクリプト内部から呼び出しが行える関数を
DLL 形式で作成することで追加が可能となります。

SocketDebugger は起動時、同一フォルダにある DLL を検索し、
Lua 拡張関数が存在するかを確認します。
対応 DLL があった場合、利用が可能であることをバージョン情報ダイアログに表示します。
また、追加された関数はスクリプトエディタ中で紫色で表示されます。

```
-----------------------------------------------
-- 送信ﾎﾞﾀﾝ押下
-----------------------------------------------
function OnSendPush()
    Logput(1,'OnSendPush')
    a = FileReadEx( 'D:¥¥SAMPLE¥¥debug.txt' )
    --a = GetEditorData()
    SendData(a)
    return 0
end
```

Lua の拡張 DLL を作成するためには、Lua のライブラリが必要となります。
ここでは、SocketDebugger 本体と同じバージョンを利用して説明します。

バージョン	Lua 5.1.2
種類	Windows x86 Static Library and Includes (Visual C++ 6 Compatible)
ファイル名称	lua5_1_2_Win32_vc6_lib.zip
ダウンロード URL	http://luabinaries.luaforge.net/download.html

Lua による処理を記述するには、C 言語、Lua 言語の多少の知識が必要となります。
ですが、公開しているサンプルを活用することで、C 言語の知識のある人であれば
Lua の記述方法は理解できるものと思います。

サンプルソースの中には、SocketDebugger から認識するための関数と
Lua の定型的な処理の関数、および参考として欲しい拡張関数の記述があります。

スクリプトから呼び出す Lua 関数の C 言語表現は常にパラメータはひとつとなります。
その、唯一の Lua 識別子から、引数などの判断を行います。

利用可能な関数や、記述方法についての詳細は Lua のリファレンスを参照ください。

具体的な例は、サンプルを SocketDebugger ホームページより
ダウンロードして確認ください。

改訂履歴		
2005年04月10日	初版作成	Ver 1.09
2005年05月28日	アプリケーションバージョンアップのため改訂	Ver 1.10
2006年02月11日	アプリケーションバージョンアップのため改訂	Ver 1.11
2007年09月09日	アプリケーションバージョンアップのため改訂	Ver 1.13
2015年07月06日	権利移行に伴い改訂 Ver.1.14～1.17の内容を追記	Ver.1.17

作成者
株式会社ユードム SocketDebugger担当一同
サポートＵＲＬ　http://sdg.udom.co.jp
e-mail sdg@udom.co.jp